Marianne Winslett (Ed.)

LNCS 5566

# Scientific and Statistical Database Management

21st International Conference, SSDBM 2009
New Orleans, LA, USA, June 2009
Proceedings

Springer

Marianne Winslett (Ed.)

# Scientific and Statistical Database Management

21st International Conference, SSDBM 2009
New Orleans, LA, USA, June 2-4, 2009
Proceedings

Springer

Volume Editor

Marianne Winslett
University of Illinois at Urbana-Champaign
Department of Computer Science
201 North Goodwin Avenue, Urbana, IL 61801, USA
E-mail: winslett@cs.uiuc.edu

Library of Congress Control Number: Applied for

CR Subject Classification (1998): H.2.8, H.2.5, H.3, H.4, J.1-3, I.3.7

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

| | |
|---|---|
| ISSN | 0302-9743 |
| ISBN-10 | 3-642-02278-2 Springer Berlin Heidelberg New York |
| ISBN-13 | 978-3-642-02278-4 Springer Berlin Heidelberg New York |

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12696451 06/3180 5 4 3 2 1 0

# Preface

This volume contains the proceedings of SSDBM 2009, the 21st International Conference on Scientific and Statistical Database Management. SSDBM 2009 took place during June 2–4, 2009, at the Hotel Monteleone in New Orleans, USA. The SSDBM conference series brings together scientific domain experts, database researchers, practitioners, and developers for the presentation and exchange of current research concepts, tools, and techniques for scientific and statistical database applications. SSDBM organizers strive to provide a stimulating environment to encourage discussion, fellowship, and exchange of ideas in all aspects of research related to scientific and statistical databases, including both original research contributions and insights from practical system design, implementation, and evaluation.

SSDBM 2009 received 76 submissions from 18 countries. Each submission was reviewed by three Program Committee members, leading to the acceptance of 29 long papers and 12 short papers. The short papers include a mix of demonstrations, poster papers, and traditional conference presentations. This year we had the goal of increasing our acceptance rate while maintaining or increasing the quality of our papers; to this end, 17 of our accepted papers were shepherded.

This year we also benefitted from three invited talks. Our keynote presentation was from Kate Keahey of Argonne National Laboratory, who talked about scientific computing on cloud platforms. Bertram Ludaescher from the University of California Davis explained what makes scientific workflows scientific, and Arie Shoshani gave an overview of new technology developed at the Scientific Data Management Center at Lawrence Berkeley National Laboratory for exploring scientific datasets.

Organizing SSDBM 2009 was a team effort that involved many people. I thank our General Chairs Mahdi Abdelguerfi and Shengru Tu for their careful attention to so many details, and the SSDBM Steering Committee for their guidance. The Program Committee and our external referees did an excellent job with their timely review and careful discussion of all our submissions. I thank our 17 shepherds for their extra effort to ensure that SSDBM remains a premier forum. I also appreciate EasyChair's great facilities for assembling the camera-ready version of the proceedings, and Ragib Hasan's help in assembling the proceedings. We are also grateful to our conference sponsors, who provided financial support for SSDBM 2009: Louisiana Technology Group (LATG), Diamond Data Systems (DDS), Sun Microsystems, NOVACES, and the Department of Computer Science at the University of New Orleans.

I hope that you enjoy the proceedings!

April 2009 Marianne Winslett

# Organization

## Program Chair

| | |
|---|---|
| Marianne Winslett | University of Illinois at Urbana-Champaign, USA |

## General Chair

| | |
|---|---|
| Mahdi Abdelguerfi | University of New Orleans, USA |

## General Co-chair

| | |
|---|---|
| Shengru Tu | University of New Orleans, USA |

## Program Committee

| | |
|---|---|
| Walid Aref | Purdue University, USA |
| Ken Barker | University of Calgary, Canada |
| Randal Burns | Johns Hopkins University, USA |
| Sarah Cohen-Boulakia | University of Paris-Sud XI, France |
| Nilesh Dalvi | Yahoo!, USA |
| Amr El Abbadi | UC Santa Barbara, USA |
| Hakan Ferhatosmanoglu | Ohio State, USA |
| Minos Garofalakis | Yahoo! & UC Berkeley, USA |
| Dimitrios Gunopolis | UC Riverside, USA |
| Amarnath Gupta | UC San Diego, USA |
| H. V. Jagadish | University of Michigan, USA |
| George Kollios | Boston University, USA |
| Yunyao Li | IBM, USA |
| Xiaosong Ma | North Carolina State University, USA |
| Claudia Medeiros | University of Campinas, Brazil |
| Jignesh M. Patel | University of Michigan, USA |
| Sunil Prabhakar | Purdue University, USA |
| Louiqa Raschid | University of Maryland, USA |
| Rajeev Rastogi | Yahoo!, USA |
| Rishi Sinha | Microsoft, USA |
| Divesh Srivastava | AT&T Labs–Research, USA |
| Alex Szalay | Johns Hopkins University, USA |
| Kian-Lee Tan | National University of Singapore, Singapore |
| Anthony Tung | National University of Singapore, Singapore |
| Kesheng Wu | Lawrence Berkeley National Laboratory, USA |
| Cong Yu | Yahoo!, USA |
| Xiaofang Zhou | University of Queensland, Australia |

## Steering Committee

| | |
|---|---|
| Arie Shoshani (Chair) | Lawrence Berkeley National Laboratory, USA |
| Moustafa A. Hammad | University of Calgary, Canada |
| Nick Koudas | University of Toronto, Canada |
| Bertram Ludaescher | University of California at Davis, USA |
| Nikos Mamoulis | University of Hong Kong, Hong Kong, China |

## Sponsors

LATG
Innovative & Effective Technology

dds™
DIAMOND DATA SYSTEMS

Sun® microsystems

NOVACES®
Advancing Process Improvement.

uno
UNIVERSITY of NEW ORLEANS

## External Reviewers

# Table of Contents

## 1. Invited Presentation

The Scientific Data Management Center: Providing Technologies for Large Scale Scientific Exploration . . . . 1
*Arie Shoshani*

## 2. Improving the End-User Experience

Query Recommendations for Interactive Database Exploration . . . . 3
*Gloria Chatzopoulou, Magdalini Eirinaki, and Neoklis Polyzotis*

Scientific Mashups: Runtime-Configurable Data Product Ensembles . . . . 19
*Bill Howe, Harrison Green-Fishback, and David Maier*

View Discovery in OLAP Databases through Statistical Combinatorial Optimization . . . . 37
*Cliff Joslyn, John Burke, Terence Critchlow, Nick Hengartner, and Emilie Hogan*

Designing a Geo-scientific Request Language - A Database Approach . . . 56
*Peter Baumann*

SEEDEEP: A System for Exploring and Querying Scientific Deep Web Data Sources . . . . 74
*Fan Wang and Gagan Agrawal*

Expressing OLAP Preferences . . . . 83
*Matteo Golfarelli and Stefano Rizzi*

## 3. Indexing, Physical Design, and Energy

Energy Smart Management of Scientific Data . . . . 92
*Ekow Otoo, Doron Rotem, and Shih-Chiang Tsao*

Data Parallel Bin-Based Indexing for Answering Queries on Multi-core Architectures . . . . 110
*Luke J. Gosink, Kesheng Wu, E. Wes Bethel, John D. Owens, and Kenneth I. Joy*

Finding Regions of Interest in Large Scientific Datasets . . . . 130
*Rishi Rakesh Sinha, Marianne Winslett, and Kesheng Wu*

Adaptive Physical Design for Curated Archives . . . . . . . . . . . . . . . . . . . . . . 148
*Tanu Malik, Xiaodan Wang, Debabrata Dash, Amitabh Chaudhary, Anastasia Ailamaki, and Randal Burns*

MLR-Index: An Index Structure for Fast and Scalable Similarity Search in High Dimensions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 167
*Rahul Malik, Sangkyum Kim, Xin Jin, Chandrasekar Ramachandran, Jiawei Han, Indranil Gupta, and Klara Nahrstedt*

## 4. Application Experience

B-Fabric: An Open Source Life Sciences Data Management System . . . . . 185
*Can Türker, Fuat Akal, Dieter Joho, and Ralph Schlapbach*

Design and Implementation of Metadata System in PetaShare . . . . . . . . . 191
*Xinqi Wang and Tevfik Kosar*

Covariant Evolutionary Event Analysis for Base Interaction Prediction Using a Relational Database Management System for RNA . . . . . . . . . . . 200
*Weijia Xu, Stuart Ozer, and Robin R. Gutell*

## 5. Invited Presentation

What Makes Scientific Workflows Scientific? . . . . . . . . . . . . . . . . . . . . . . . . . 217
*Bertram Ludäscher*

## 6. Workflow

Enabling Ad Hoc Queries over Low-Level Scientific Data Sets . . . . . . . . . . 218
*David Chiu and Gagan Agrawal*

Exploring Scientific Workflow Provenance Using Hybrid Queries over Nested Data and Lineage Graphs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 237
*Manish Kumar Anand, Shawn Bowers, Timothy McPhillips, and Bertram Ludäscher*

Data Integration with the DaltOn Framework – A Case Study . . . . . . . . . 255
*Stefan Jablonski, Bernhard Volz, M. Abdul Rehman, Oliver Archner, and Olivier Curé*

Experiment Line: Software Reuse in Scientific Workflows . . . . . . . . . . . . . . 264
*Eduardo Ogasawara, Carlos Paulino, Leonardo Murta, Cláudia Werner, and Marta Mattoso*

Tracking Files in the Kepler Provenance Framework . . . . . . . . . . . . . . . . . . 273
*Pierre Mouallem, Roselyne Barreto, Scott Klasky, Norbert Podhorszki, and Mladen Vouk*

BioBrowsing: Making the Most of the Data Available in Entrez ........ 283
*Sarah Cohen-Boulakia and Kevin Masini*

Using Workflow Medleys to Streamline Exploratory Tasks ............. 292
*Emanuele Santos, David Koop, Huy T. Vo, Erik W. Anderson, Juliana Freire, and Cláudio Silva*

## 7. Query Processing

Experiences on Processing Spatial Data with MapReduce ............. 302
*Ariel Cary, Zhengguo Sun, Vagelis Hristidis, and Naphtali Rishe*

Optimization and Execution of Complex Scientific Queries over Uncorrelated Experimental Data ................................ 320
*Ruslan Fomkin and Tore Risch*

Comprehensive Optimization of Declarative Sensor Network Queries .... 339
*Ixent Galpin, Christian Y.A. Brenninkmeijer, Farhana Jabeen, Alvaro A.A. Fernandes, and Norman W. Paton*

Efficient Evaluation of Generalized Tree-Pattern Queries with Same-Path Constraints ................................................ 361
*Xiaoying Wu, Dimitri Theodoratos, Stefanos Souldatos, Theodore Dalamagas, and Timos Sellis*

Mode Aware Stream Query Processing .............................. 380
*Mingrui Wei and Elke Rundensteiner*

Evaluating Reachability Queries over Path Collections ............... 398
*Panagiotis Bouros, Spiros Skiadopoulos, Theodore Dalamagas, Dimitris Sacharidis, and Timos Sellis*

## 8. Similarity Search

Easing the Dimensionality Curse by Stretching Metric Spaces .......... 417
*Ives R.V. Pola, Agma J.M. Traina, and Caetano Traina Jr.*

Probabilistic Similarity Search for Uncertain Time Series ............. 435
*Johannes Aßfalg, Hans-Peter Kriegel, Peer Kröger, and Matthias Renz*

Reverse k-Nearest Neighbor Search Based on Aggregate Point Access Methods ........................................................ 444
*Hans-Peter Kriegel, Peer Kröger, Matthias Renz, Andreas Züfle, and Alexander Katzdobler*

Finding Structural Similarity in Time Series Data Using Bag-of-Patterns Representation ................................................ 461
*Jessica Lin and Yuan Li*

## 9. Keynote Address

Cloud Computing for Science . . . . . . . . . . 478
*Kate Keahey*

## 10. Mining

Classification with Unknown Classes . . . . . . . . . . 479
*Chetan Gupta, Song Wang, Umeshwar Dayal, and Abhay Mehta*

HSM: Heterogeneous Subspace Mining in High Dimensional Data . . . . . . 497
*Emmanuel Müller, Ira Assent, and Thomas Seidl*

Split-Order Distance for Clustering and Classification Hierarchies . . . . . . 517
*Qi Zhang, Eric Yi Liu, Abhishek Sarkar, and Wei Wang*

Combining Multiple Interrelated Streams for Incremental Clustering . . . . 535
*Zaigham Faraz Siddiqui and Myra Spiliopoulou*

Improving Relation Extraction by Exploiting Properties of the Target Relation . . . . . . . . . . 553
*Eric Normand, Kevin Grant, Elias Ioup, and John Sample*

*Cor-Split*: Defending Privacy in Data Re-publication from Historical Correlations and Compromised Tuples . . . . . . . . . . 562
*Daniele Riboni and Claudio Bettini*

A Bipartite Graph Framework for Summarizing High-Dimensional Binary, Categorical and Numeric Data . . . . . . . . . . 580
*Guanhua Chen, Xiuli Ma, Dongqing Yang, Shiwei Tang, and Meng Shuai*

## 11. Spatial Data

Region Extraction and Verification for Spatial and Spatio-temporal Databases . . . . . . . . . . 598
*Mark McKenney*

Identifying the Most Endangered Objects from Spatial Datasets . . . . . . . 608
*Hua Lu and Man Lung Yiu*

Constraint-Based Learning of Distance Functions for Object Trajectories . . . . . . . . . . 627
*Wei Yu and Michael Gertz*

**Author Index** . . . . . . . . . . 647

# The Scientific Data Management Center: Providing Technologies for Large Scale Scientific Exploration

Arie Shoshani

Lawrence Berkeley National Laboratory
shoshani@lbl.gov

Managing scientific data has been identified by the scientific community as one of the most important emerging needs because of the sheer volume and increasing complexity of data being collected. Effectively generating, managing, and analyzing this information requires a comprehensive, end-to-end approach to data management that encompasses all of the stages from the initial data acquisition to the final analysis of the data. Fortunately, the data management problems encountered by most scientific domains are common enough to be addressed through shared technology solutions. Based on community input, the SDM center has identified three significant requirements. First, more efficient access to storage systems is needed. In particular, parallel file system and I/O system improvements are needed to write and read large volumes of data without slowing a simulation, analysis, or visualization engine. These processes are complicated by the fact that scientific data are structured differently for specific application domains, and are stored in specialized file formats. Second, scientists require technologies to facilitate better understanding of their data, in particular the ability to effectively perform complex data analysis and searches over extremely large data sets. Specialized feature discovery and statistical analysis techniques are needed before the data can be understood or visualized. Furthermore, interactive analysis requires indexing techniques for efficiently searching and selecting subsets of interest are needed. Finally, generating the data, collecting and storing the results, keeping track of data provenance, data post-processing, and analysis of results is a tedious, fragmented process. Workflow tools for automation of this process in a robust, tractable, and recoverable fashion are required to enhance scientific exploration.

Over the last several years, the technologies developed by the SDM center, which is supported by the Department of Energy in the U.S. have been deployed in a variety of application domains. Some of the technologies that will be described in this talk include:

– More than a tenfold speedup in writing and reading netCDF files has been achieved by developing MPI-IO based Parallel netCDF software being utilized by astrophysics, climate, and fusion scientists.
– A method for the correct classification of orbits in puncture plots from the National Compact Stellarator eXperiment (NCSX) at PPPL was developed

by converting the data into polar coordinates and fitting second-order polynomials to the data points.
– A new bitmap indexing method has enabled efficient search over billions of collisions (events) in High Energy Physics, and is being applied to combustion, astrophysics, and visualization domains. It achieves more than a tenfold speedup relative to ant known indexing methods.
– The development of a Parallel R, an open source parallel version of the popular statistical package R. These are being applied to climate, GIS, and mass spec proteomics applications.
– An easy-to-use GUI-based software, called ProRata, has provided Biology researchers with robust quantification of protein abundance in high-throughput shotgun proteomics data.
– A scientific workflow management and execution system (called Kepler) has been developed and deployed within multiple scientific domains, including genomics and astrophysics. The system supports design and the execution of flexible and reusable, component-oriented workflows.

# Query Recommendations for Interactive Database Exploration

Gloria Chatzopoulou[1,⋆], Magdalini Eirinaki[2], and Neoklis Polyzotis[3]

[1] Computer Science Dept., University of California Riverside, USA
chatzopd@cs.ucr.edu
[2] Computer Engineering Dept., San Jose State University, USA
magdalini.eirinaki@sjsu.edu
[3] Computer Science Dept., University of California Santa Cruz, USA
alkis@cs.ucsc.edu

**Abstract.** Relational database systems are becoming increasingly popular in the scientific community to support the interactive exploration of large volumes of data. In this scenario, users employ a query interface (typically, a web-based client) to issue a series of SQL queries that aim to analyze the data and mine it for interesting information. First-time users, however, may not have the necessary knowledge to know where to start their exploration. Other times, users may simply overlook queries that retrieve important information. To assist users in this context, we draw inspiration from Web recommender systems and propose the use of personalized query recommendations. The idea is to track the querying behavior of each user, identify which parts of the database may be of interest for the corresponding data analysis task, and recommend queries that retrieve relevant data. We discuss the main challenges in this novel application of recommendation systems, and outline a possible solution based on collaborative filtering. Preliminary experimental results on real user traces demonstrate that our framework can generate effective query recommendations.

## 1 Introduction

Relational database systems are becoming increasingly popular in the scientific community to manage large volumes of experimental data. Examples include the Genome browser[1] that provides access to a genomic database, and Sky Server[2] that stores large volumes of astronomical measurements. The main advantage of a relational database system is that it supports the efficient execution of complex queries, thus enabling users to interactively explore the data and retrieve interesting information. It should be noted that the aforementioned systems employ web-based query interfaces in order to be accessible to a broad user base.

⋆ This work was performed while the author was affiliated with UC Santa Cruz.
[1] http://genome.ucsc.edu/
[2] http://cas.sdss.org/

Even though a database system offers the means to run complex queries over large data sets, the discovery of useful information remains a big challenge. Users who are not familiar with the database may overlook queries that retrieve interesting data, or they may not know what parts of the database provide useful information. This issue clearly hinders data exploration, and thus reduces the benefits of using a database system.

To address this important problem, we draw inspiration from the successful application of recommender systems in the exploration of Web data. In particular, we focus on approaches based on user-based collaborative filtering. The premise is simple: If a user A has similar querying behavior to user B, then they are likely interested in the same data. Hence, the queries of user B can serve as a guide for user A.

The transfer of this paradigm entails several challenging problems. In web collaborative filtering systems, a user-item matrix approach is used to generate recommendations. More specifically, each user is represented as an item vector, where the values of the vector elements correspond to the user's preferences for each item (such as movie ratings, purchased products, read articles, etc.) The similarities between users in this representation can be easily computed using vector similarity metrics. Given the most similar users and their preferences, the collaborative filtering system can subsequently predict what items will interest each user, and generate item recommendations.

The aforementioned methodology cannot be directly applied to the context of a relational database for several reasons. First, we observe that in the case of a database the "items" of interest are database records, and the users access these items indirectly by posing SQL queries. Thus, even though the users' behavior is identified by the set of queries they send to the database, their interest lies on the database tuples they retrieve. Given that SQL is a declarative language, however, the same data can be retrieved in more than one way. This complicates the evaluation of similarity among users based on their queries alone, since it is no longer obvious whether they are interested in the same "items".

This raises a second important issue that needs some consideration. The similarity between users can be expressed as the similarity between the fragments of their queries or, alternatively, the data that they retrieve. This is not as straightforward, since a query fragment or a tuple might have different levels of importance in different user sessions. Thus, we must be able to create implicit user profiles that model those levels of importance, in order to effectively compare the users.

Finally, contrary to the user-based collaborative filtering approach, the recommendation to the users have to be in the form of SQL queries, since those actually describe what the retrieved data represent. Thus, we need to "close the loop" by first decomposing the user queries into lower-level components in order to compute similarities and make predictions, and then re-construct them back to SQL queries in order to recommend them. Moreover, these SQL queries must be meaningful and intuitive, so that users can parse them and understand their intent and usefulness. All those issues make the problem of interactive database exploration very different from its web counterpart.

In this paper, we present our work in the development of a query recommender system for relational databases. We first discuss an abstract framework that conceptualizes the problem and defines specific components that must be instantiated in order to develop a solution. Based on this framework, we develop a solution that transfers the paradigm of collaborative filtering in the context of relational queries. The recommended solution can be implemented using existing technology, and is thus attractive for a real-world deployment. Finally, we present an experimental study on real user traces from the Sky Server database. Our results indicate that our first-cut solution can provide effective query recommendations, and thus demonstrate the potential of our approach in practice.

The remainder of the paper is structured as follows. We review the related work in Section 2 and cover some preliminaries in Section 3. Section 4 discusses the conceptual framework and its instantiation. The experimental results are presented in Section 5. Section 6 concludes the paper and outlines directions for future work.

## 2 Related Work

So far, the work that has been done in the area of personalized databases has focused to keyword-based query recommendation systems [1]. In this scenario, a user can interact with a relational database through a web interface that allows him/her to submit keywords and retrieve relevant content. The personalization process is based on the user's keyword queries, those of previous users, as well as an explicit user profile that records the user's preferences with regards to the content of the database. Clearly, our approach is different from this scenario in several ways. First, the proposed framework is meant to assist users who pose complex SQL queries to relational databases. Moreover, the system does not require from its users to create an explicit profile. This gives a higher level of flexibility to the system, since the same user might have different information needs during different explorations of the database.

Our inspiration draws from the successful application of user-based collaborative filtering techniques, proposed in the Web context [2, 3, 4, 5, 6, 7, 8, 9, 10]. As previously mentioned, this approach cannot be directly applied to the relational database context. The inherent nature of interactive database exploration raises certain implications that cannot be addressed by the straightforward collaborative filtering approach. In this work, we are based on its premises, but extend them in order to apply them in the database environment.

The challenges of applying data mining techniques to the database query logs are also addressed in [11]. In this work, the authors outline the architecture of a Collaborative Query Management System targeted at large-scale, shared-data environments. As part of this architecture, they independently suggest that data mining techniques, such as clustering or association rules, can be applied to the query logs in order to provide the users with query suggestions. We should stress, however, that contrary to our work, the authors do not provide any technical details on how such a recommendation system could be implemented.

The work presented in this paper is part of the QueRIE (Query Recommendations for Interactive database Exploration) project. In this project, we investigate the application of personalization techniques in interactive database exploration, particularly to assist the user in discovering interesting subsets of the database with minimal effort.

## 3 Preliminaries

Our work considers the interactive exploration of a relational database using SQL queries. In what follows, we summarize some basic notions in this context that will be used in the remainder of the paper.

### 3.1 Database and Querying Model

We consider a relational database comprising $N$ relations denoted as $R_1, \ldots, R_N$. We use $Q$ to denote a Select-Project-Join (SPJ) query over the database, and $ans(Q)$ for its result set. We focus on the class of SPJ queries because they are common in interactive database exploration, particularly among the group of novice users which is the focus of our work.

We say that a tuple $\tau$ of some relation $R_n$, $1 \le n \le N$, is a witness for a query $Q$ if $\tau$ contributes to least one result in $ans(Q)$. We use $R_n^Q$ to denote the set of witnesses for $Q$ from relation $R_n$. (For notational convenience, we assume that $R_n^Q = \emptyset$ if relation $R_n$ is not mentioned in $Q$.) Overall, the subsets $R_1^Q, \ldots, R_N^Q$ contain the tuples that are used to generate the results of $Q$. In that respect, we say that $R_1^Q, \ldots, R_N^Q$ is the subset of the database touched by $Q$.

### 3.2 Interactive Data Exploration

Users typically explore a relational database through a sequence of SQL queries. The goal of the exploration is to discover interesting information or verify a particular hypothesis. The queries are formulated based on this goal and reflect the user's overall information need. As a consequence, the queries posted by a user during one "visit" (commonly called *session*) to the database are typically correlated in that the user formulates the next query in the sequence after having inspected the results of previous queries.

We identify users with unique integer identifiers. Given a user $i$, let $\mathcal{Q}_i$ denote the set of SQL queries that the user has posed. In accordance with the previous definitions, we assume that the SQL queries belong to the class of SPJ queries. We define $R_n^i$, $1 \le n \le N$ as the union of $R_n^Q$ for $Q \in \mathcal{Q}_i$, i.e., the set of tuples of relation $R_n$ that the user's queries have touched. Hence, $R_1^i, \ldots, R_N^i$ represent the subset of the database that has been accessed by user $i$. A summary of the notation used throughout this paper is included in Table 1.

**Table 1.** Notation Summary

| | |
|---|---|
| $R_n$ | Relation $n$ |
| $R_n^Q$ | Set of witnesses for query $Q$ from $R_j$ |
| $R_n^i$ | Set of tuples of $R_n$ that queries of user $i$ have retrieved |
| $S_i$ | Session summary of user $i$ |
| $S_0^{\text{pred}}$ | Extended session summary for current user |

## 4 Personalized Query Recommendations

The problem of personalized query recommendations can be formulated as follows: Given a user that is currently exploring the database, recommend queries that might be of interest to him/her. To generate such recommendations, the system will rely on information gathered from the querying behavior of past users, as well as the queries posed by the current user so far.

The information flow of the QueRIE framework is shown in Figure 1. The active user's queries are forwarded to both the DBMS and the Recommendation Engine. The DBMS processes each query and returns a set of results. At the same time, the query is stored in the Query Log. The Recommendation Engine combines the current user's input with information gathered from the database interactions of past users, as recorded in the Query Log, and generates a set of query recommendations that are returned to the user.

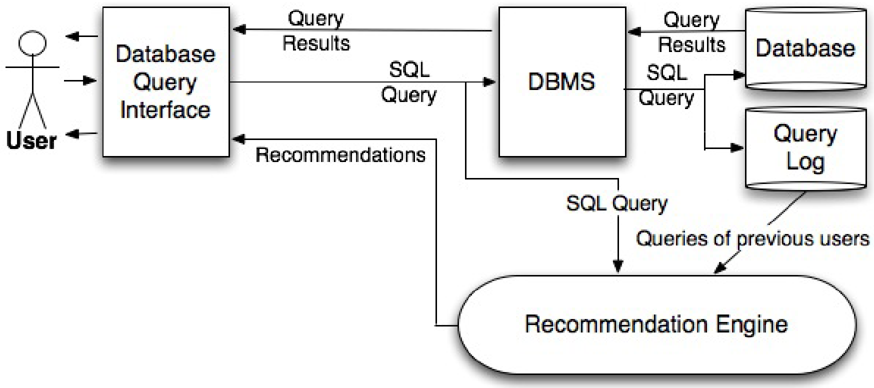


**Fig. 1.** QueRIE Architecture

In what follows, we identify the current user with the id 0, and note that $\mathcal{Q}_0$ contains the queries that the user has posed thus far. We use $\{1, \ldots, h\}$ to denote the set of past users based on which recommendations are generated.

The following sections describe a solution to this interesting problem of generating personalized query recommendations. We begin by discussing a conceptual framework that can encompass different approaches, and then propose an instantiation based on collaborative filtering.

### 4.1 Conceptual Framework

Clearly, the queries of each user touch a subset of the database that is relevant for the analysis the user wants to perform. We assume that this subset is modeled as a *session summary.* This summary captures the parts of the database accessed by the user and incorporates a metric of importance for each part. For instance, a crude summary may contain the names of the relations that appear in the queries of the user, and the importance of each relation can be measured as the number of queries that reference it. On the other extreme, a detailed summary may contain the actual results inspected by the user, along with an explicit rating of each result tuple. Assuming that the choice of the summary is fixed for all users, we use $S_i$ to denote the summary for user $i$.

To generate recommendations, our framework generates a "predicted" summary $S_0^{\text{pred}}$. This summary captures the predicted degree of interest of the active user with respect to all the parts of the database, including those that the user has not explored yet, and thus serves as the seed for the generation of recommendations. As an example, if the summary $S_0$ contains the names of the relations that the user has referenced so far, then $S_0^{\text{pred}}$ may contain more relations that might be of interest, along with the respective degree of "interestingness" for each part.

Using $S_0^{\text{pred}}$, the framework constructs queries that cover the subset of the database with the highest predicted importance. In turn, these queries are presented to the user as recommendations.

Overall, our framework consists of three components: (a) the construction of a session summary for each user $i$ based on the queries in $\mathcal{Q}_i$, (b) the computation of $S_0^{\text{pred}}$ based on the active user $S_0$ and the summaries $S_1, \ldots, S_h$ of past users, and (c) the generation of queries based on $S_0^{\text{pred}}$. An interesting point is that components (a) and (c) form a closed loop, going from queries to summaries and back. This is a conscious design choice following the fact that all user interaction with a relational database occurs through declarative queries.

### 4.2 A Witness-Based Collaborative Filtering Approach

We now discuss an instantiation of the previously described framework. In the following sections, we discuss the model and construction of session summaries using witnesses, the computation of the extended summary $S_0^{\text{pred}}$, and the recommendation algorithm.

**Session Summaries.** The session summary $S_i$ is represented as a weighted vector that corresponds to the database tuples. We assume that the total number of tuples in the database, and as a consequence the length of the vector, is $T$. The weight $S_i[\tau]$ represents the importance of a given tuple $\tau \in T$ in session $S_i$. In what follows, we describe the computation of tuple weights in $S_i$.

We assume that the vector $S_Q$ represents a single query $Q \in \mathcal{Q}_i$. The value of each element $S_Q[\tau]$ signifies the importance of the tuple as the witness for $Q$. We propose two different weighting schemes for computing the tuple weights in $S_i$:

*Binary weighting scheme.*

$$S_Q[\tau] = \begin{cases} 1 & \text{if } \tau \text{ is a witness;} \\ 0 & \text{if } \tau \text{ is not a witness.} \end{cases} \tag{1}$$

This is the most straightforward approach. There are two options: either a tuple is a witness in $Q$, or not. All participating tuples receive the same importance weight.

*Result weighting scheme.*

$$S_Q[\tau] = \begin{cases} 1/|ans(Q)| & \text{if } \tau \text{ is a witness;} \\ 0 & \text{if } \tau \text{ is not a witness.} \end{cases} \tag{2}$$

Here $ans(Q)$ is the result-set of $Q$. The intuition is that the importance of $\tau$ is diminished if $Q$ returns many results, as this is an indication that the query is "unfocused". On the other hand, a small $ans(Q)$ implies that the query is very specific, and thus the witnesses have high importance.

Given the vectors $S_Q$ for $Q \in \mathcal{Q}_i$, we define the session summary of user $i$ as:

$$S_i = \sum_{Q \in \mathcal{Q}_i} S_Q. \tag{3}$$

Using the session summaries of the past users, we can construct the $(h \times T)$ *session-tuple matrix* which, as in the case of the user-item matrix in web recommender systems, will be used as input to our recommendation algorithm.

**Computing $\boldsymbol{S_0^{\text{pred}}}$.** Similarly to session summaries $S_i$, the predicted summary $S_0^{\text{pred}}$ is a vector of tuple weights. Each weight signifies the predicted importance of the corresponding tuple for the active user. In order to compute those weights, we adopt the method of a linear summation that has been successfully employed in user-based collaborative filtering. More specifically, we assume the existence of a function $sim(S_i, S_j)$ that measures the similarity between two session summaries and takes values in $[0, 1]$. The similarity function $sim(S_i, S_j)$ can be realized with any vector-based metric. In this work, we employ the cosine similarity:

$$sim(S_i, S_j) = \frac{S_i S_j}{\|S_i\|_2 \|S_j\|_2}. \tag{4}$$

This implies that two users are similar if their queries imply similar weights for the database tuples.

The predicted summary is defined as a function of the current user's summary $S_0$ and the normalized weighted sum of the existing summaries:

$$S_0^{\text{pred}} = \alpha * S_0 + (1 - \alpha) * \frac{\sum_{1 \le i \le h} sim(S_0, S_i) \cdot S_i}{\sum_{1 \le i \le h} sim(S_0, S_i)} \tag{5}$$

The value of the "mixing" factor $\alpha \in [0, 1]$ determines which users' traces will be taken into consideration when computing the predicted summary. If $\alpha = 0$, then we follow the user-based collaborative filtering approach and take into account only the past users' traces. On the other hand, when $\alpha = 1$, only the active user's session summary is taken into account when generating recommendations, resulting in what is known in the recommendation systems as content-based filtering. Finally, any value in between allows us to bias the predicted vector and assign more "importance" to either side, or equal "importance" to both the current and the previous users (when $\alpha = 0.5$). This bias can be useful for two reasons. First, we do not want to exclude from the recommendation set any queries that touch the tuples already covered by the user. Even though there might exist some overlap, such queries may provide a different, more intuitive presentation (e.g., a different PROJECT clause), making it easier for the user to search for the information she is looking for. Second, by including the covered tuples in $S_0^{\text{pred}}$, we are able to predict queries that combine the seen tuples with unseen tuples from other relations. In other words, we are able to predict queries that "expand" on the results already observed by the user. Intuitively, we expect from the active user to behave in a similar way by posing queries that cover adjacent or overlapping parts of the database, in order to locate the information they are seeking. These two observations derive from the nature of database queries and are in some sense inherent in the problem of personalized query recommendations.

**Generating Query Recommendations.** Having computed $S_0^{\text{pred}}$, the algorithm recommends queries that retrieve tuples of high predicted weights. One possibility would be to automatically synthesize queries out of the predicted tuples in $S_0^{\text{pred}}$, but this approach has an inherent technical difficulty. Another drawback of this approach is that the resulting queries may not be intuitive and easily understandable. This is important in the context of query recommendations, as users must be able to interpret the recommended queries before deciding to use them.

To avoid the aforementioned issues, we choose to generate recommendations using the queries of past users. Such recommendations are expected to be easily understandable, since they have been formulated by a human user. More concretely, we maintain a sample of the queries posed by previous users. In the context of predicting queries for the active user, we assign to each query $Q$ in the sample an "importance" with respect to $S_0^{\text{pred}}$. This importance is computed as the similarity between the query vector $S_Q$ and $S_0^{\text{pred}}$, and is defined as follows:

$$rank(Q, S_0^{\text{pred}}) = sim(S_Q, S_0^{\text{pred}}). \tag{6}$$

Hence, a query has high rank if it covers the important tuples in $S_0^{\text{pred}}$. The top ranked queries are then returned as the recommendation.

## 5 Experimental Evaluation

We completed a prototype implementation of the framework described in the previous section. We are also in the process of developing a visual query interface that employs our framework to provide on-demand recommendations to users who navigate the database. In this section we present preliminary experimental results of using our system with real database traces, as well as examples of queries and the related recommendations generated for this data set.

## 5.1 Data Set

We evaluated our framework using traces of the Sky Server database[3]. The traces contain queries posed to the database between the years 2006 and 2008. We used the methods described in [12] to clean and separate the query logs in sessions. The characteristics of the data set and the queries are summarized in Table 2.

**Table 2.** Data Set Statistics

| | |
|---|---|
| Database size | 2.6TB |
| #Sessions | 720 |
| #Queries | 6713 |
| #Distinct queries | 4037 |
| #Distinct witnesses | 13,602,430 |
| Avg. number of queries per session | 9.3 |
| Min. number of queries per session | 3 |

## 5.2 Methodology

We employ 10-fold cross validation to evaluate the proposed framework. More concretely, we partition the set of user sessions in 10 equally sized subsets, and in each run we use 9 subsets as the training set and we generate recommendations for the sessions in the remaining subset. For each test user session of size $L$, we build the session summary $S_0$ using $L-1$ queries and we thus generate recommendations for the $L$-th query of the user. In order to generate the top-$n$ recommendations we use the queries in the current training set.

We experimented with different values for $n$. In this paper we report the results for $n=3$ and $n=5$. A larger recommendation set might end up being overwhelming for the end user, who is usually interested in selecting only a few recommended queries.

The effectiveness of each recommended query is measured against the $L$-th query of the session, using the following precision and recall metrics:

$$precision = \frac{\mid \tau_{Q_L} \cap \tau_{Q_R} \mid}{\mid \tau_{Q_R} \mid} \tag{7}$$

$$recall = \frac{\mid \tau_{Q_L} \cap \tau_{Q_R} \mid}{\mid \tau_{Q_L} \mid} \tag{8}$$

where $\tau_{Q_L}$ represents the witnesses of the $L$-th query and $\tau_{Q_R}$ represents the witnesses of the recommended query. The precision metric shows the percentage

[3] We used version BestDR6.

of “interesting” tuples to the user with respect to all the recommended tuples. The recall metric captures the hit ratio of each recommended query with respect to the last query of the user.

Following the practice of previous studies in recommender systems [13], we report for each user session the maximum recall over all the recommended queries, and compute the precision for the query that achieved maximum recall. We also report the average precision and recall for one set of recommendations. Unless otherwise noted, we set $\alpha = 0.5$.

### 5.3 Results

We conducted several experiments to evaluate different aspects of our system. Overall, the results show the feasibility of query recommendations as a guide for interactive data exploration.

In the first experiment, we evaluate the effectiveness of recommendations for the two different tuple-weighting schemes described in Section 4.2, namely the *Binary* and the *Result* methods. Figures 2 and 3 show the inverse cumulative frequency distribution (inverse CFD) of the recorded precision and recall for the test sessions. (Recall that all sessions are used as test sessions, using the 10-fold cross validation methodology described earlier.) A point $(x, y)$ in this graph signifies that $x\%$ of user sessions had precision/recall $\geq y$. For instance, as shown in Figure 3, the *Binary* method achieves a perfect recall (i.e., the recommendations cover all the tuples that the user covers with his/her last query) for more than half of the test sessions. We also observe that several

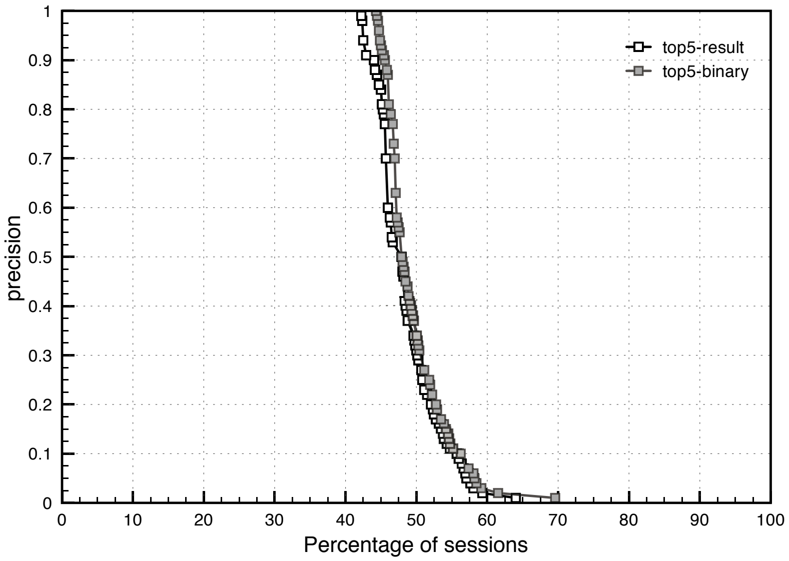


**Fig. 2.** Inverse CFD of precision for top-5 recommendations

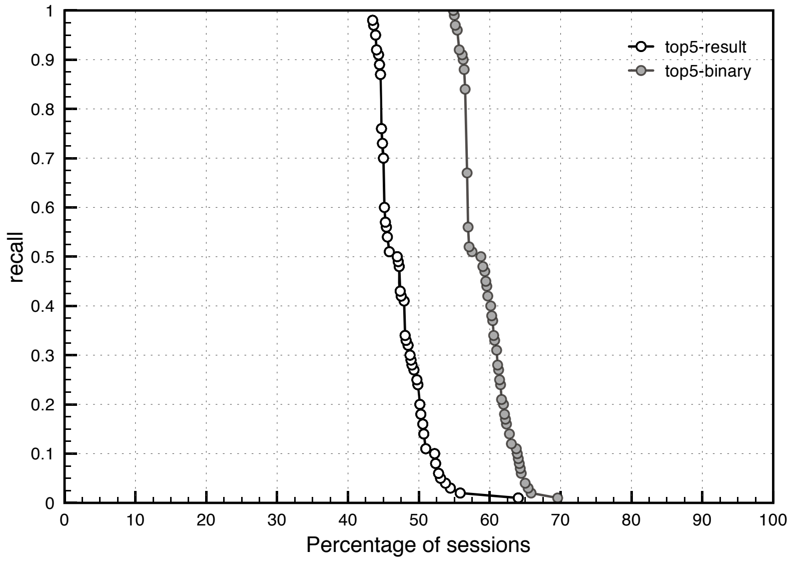


**Fig. 3.** Inverse CFD of recall for top-5 recommendations

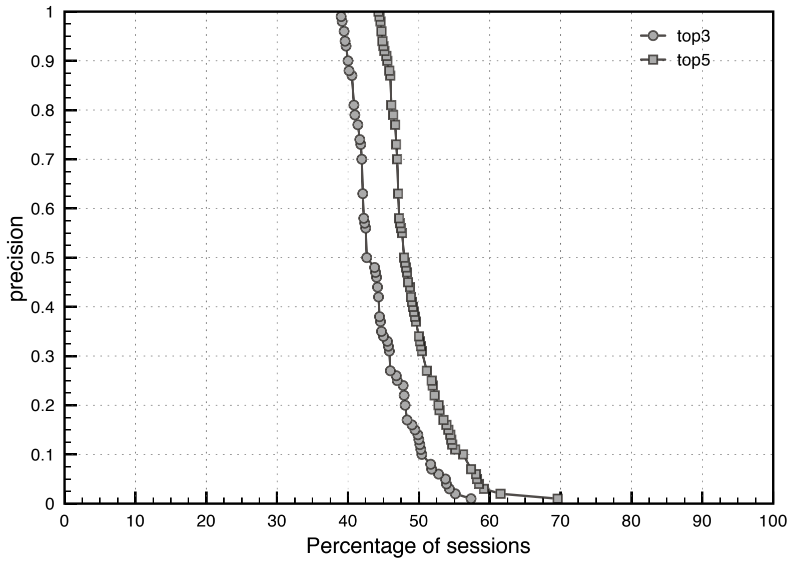


**Fig. 4.** Precision of top-3 and top-5 recommendations

test sessions have a precision and recall of 0, i.e., the recommendations did not succeed in predicting the intentions of the user. On closer inspection, these test sessions are very dissimilar to training sessions, and thus the framework fails to compute useful weights for $S_0^{\text{pred}}$.

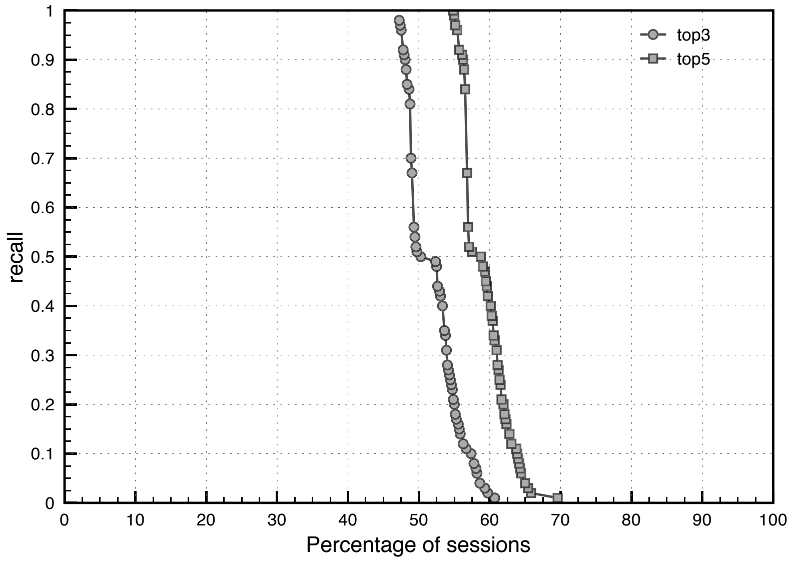


**Fig. 5.** Recall of top-3 and top-5 recommendations

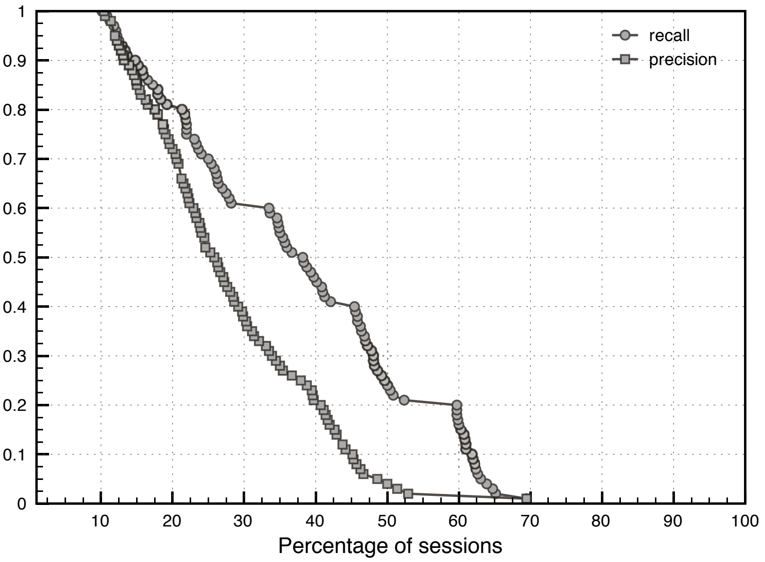


**Fig. 6.** Average precision and recall of top-5 recommendations

Overall, we observe that both methods achieve similar precision. This means that both methods' recommended queries cover the same percentage of interesting tuples for the user. The *Binary* method, however, achieves much better results than the *Result* one in terms of recall. As previously mentioned, recall represents the number of recommended tuples that were of interest to the user

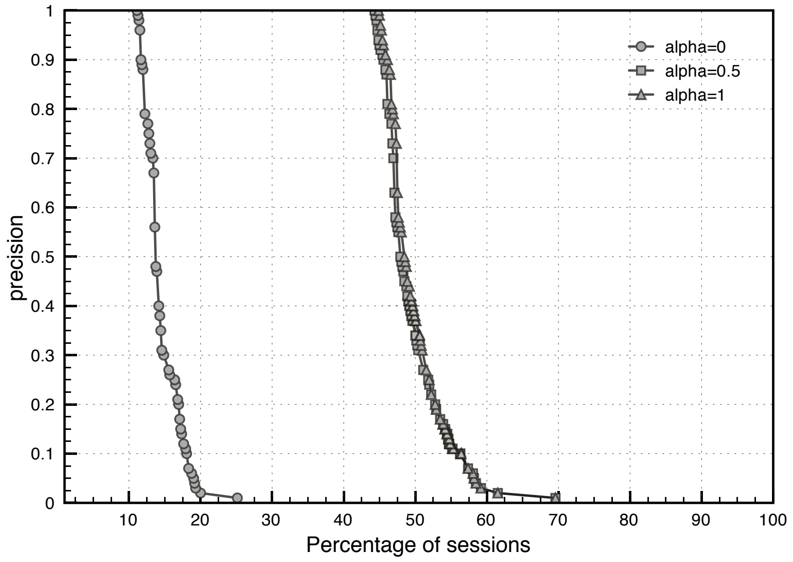


**Fig. 7.** Precision of top-5 recommendations for different $\alpha$ values

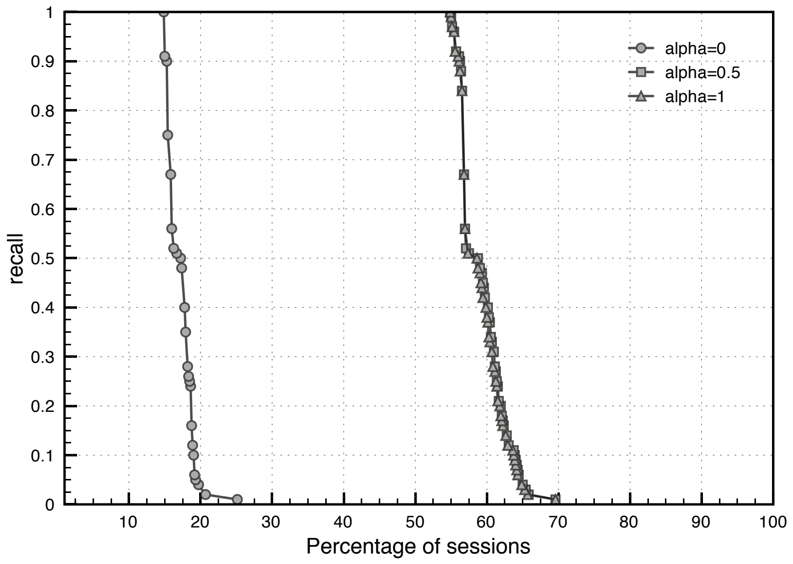


**Fig. 8.** Recall of top-5 recommendations for different $\alpha$ values

with respect to the user's last query, and is a better predictor in terms of usefulness of the recommended query. This finding implies that the result size of the query may not be a good indicator of the focus of users. Thus, in the experiments that follow, we report the results of the *Binary* weighting scheme only.

In the next set of experiments, we compare the recommendations with regards to the size of the recommendation set. More specifically, in Figures 4 and 5 we compare the top-3 and top-5 recommendation sets in terms of precision and recall respectively. Both recommendation sets achieve good results, being 100% accurate in almost half test sessions. The top-5 recommendation set seems to be performing better than the top-3 one, both in terms of precision and recall. This can be justified by the fact that we report the maximum recall over all the recommended queries and compute the precision for the query that achieved maximum recall. This query might not always be included in the top-3 ones, but is very often included in the top-5 recommendations. Notably, in about 55% of the test sessions, the maximum recall was 1, meaning that the recommended query covered all the tuples that were retrieved by the user's original one.

Figure 6 shows the average recall and precision of all top-5 recommended queries. In this case we achieve high precision and recall for almost 1/3 of the test sessions. The lower average precision and recall for the remaining sessions means that some recommended queries might not be as accurate in covering the interesting subsets of the database, dragging the overall average down. In real-life applications, however, it is likely that the active user will be able to select the few recommendations closest to his/her interests. This motivates the use of the maximum recall metric, which is used in the experiments of Figures 4 and 5.

Next, we evaluate the effect of the mixing factor $\alpha$ (Equation 5). More specifically, we evaluate the precision and recall of the top-5 recommendations for the pure user-based collaborative filtering approach ($\alpha = 0$), the content-based filtering approach ($\alpha = 1$), as well as the case when both inputs are given equal importance ($\alpha = 0.5$). As shown in Figures 7 and 8, the pure collaborative filtering approach ($\alpha = 0$) yields worst results with respect to the other two approaches, in terms of both the precision and the recall of the recommendations. The comparison of the other two approaches ($\alpha = 0.5$ and $\alpha = 1$) shows that the combination of both sources ($\alpha = 0.5$) yields slightly better results in terms of recall. We should point out, however, that the results shown here are tightly connected to the specific data set and workload. In practice, we expect that $\alpha$ will be calibrated prior to deploying the recommendation algorithm, based on the characteristics of the database and a representative user workload.

Finally, we present some examples of recommended queries for sessions in which the recommendations were 100% successful in terms of maximum recall and precision. Table 3 shows a description of the session's characteristics and the recommended query. The table lists recommendations for three user sessions, where the users had a very different progression in terms of the submitted queries. Our system was able to recommend a query that returned exactly the same results as the actual last query of the user, without the two queries being necessarily identical. This evidence demonstrates the usefulness of our approach in assisting users to interactively explore a relational database.

**Table 3.** Query recommendations examples

| **Session description** | **Recommended query** |
|---|---|
| Each consecutive query was posted to a different table. | SELECT * <br> FROM PhotoZ <br> WHERE objId = 0x082802f0c19a003e; |
| The user kept refining the same query adding exactly one selection predicate in every consecutive query. | SELECT p.ra, p.dec, s.z, s.ew, s.ewErr <br> FROM specLine s, PhotoObj p <br> WHERE s.specObjId = p.specObjid AND s.specLineId = 1549; |
| The user posted queries to the same tables, but each query had several selection clauses in addition to the previous one. | SELECT top 10 L1.height Halpha_h, L2.height Hbeta_h, <br> L3.height OIII_h, L4.height NII_h, L1.sigma Halpha_sig, <br> L2.sigma Hbeta_sig, L3.sigma OIII_sig, L4.sigma NII_sig <br> FROM Specline L1, Specline L2, Specline L3, Specline L4, SpecObj <br> WHERE SpecObj.SpecObjID = L1.SpecObjID AND <br> SpecObj.SpecObjID = L2.SpecObjID AND <br> SpecObj.SpecObjID = L3.SpecObjID AND <br> SpecObj.SpecObjID = L4.SpecObjID AND <br> SpecObj.specClass = 3 AND <br> L1.lineID = 6565 AND <br> L2.lineID = 4863 AND <br> and L3.lineID = 5008 AND <br> L4.lineID = 6585; |

## 6 Conclusions

In this paper, we present a query recommendation framework supporting the interactive exploration of relational databases and an instantiation of this framework based on user-based collaborative filtering. The experimental evaluation demonstrates the potential of the proposed approach.

We should stress that this is a first-cut solution to the very interesting problem of personalized query recommendations. There are many open issues that need to be addressed. For instance, an interesting problem is that of identifying "similar" queries in terms of their structure and not the tuples they retrieve. Two queries might be semantically similar but retrieve different results due to some filtering conditions. Such queries need to be considered in the recommendation process. We are currently working on extending our framework to cover such query similarities. Another interesting direction is to apply item-based collaborative filtering instead of the user-based approach of the current framework. We also intend to explore other approaches for instantiating the proposed conceptual framework.

We are also in the process of developing a visual query interface for the QueRIE system and plan to evaluate its performance using real users. To ensure that the system generates real-time recommendations for the active users of a database, we need to devise smart methods to compress the session-tuple matrix and to speed up the computation of similarities. In this direction, we plan to leverage randomized sketching techniques as a compression method [14, 15, 16].

## References

[1] Koutrika, G., Ioannidis, Y.: Personalized queries under a generalized preference model. In: ICDE 2005: Proceedings of the 21st International Conference on Data Engineering, pp. 841–852 (2005)

[2] Adomavicius, G., Kwon, Y.: New recommendation techniques for multicriteria rating systems. IEEE Intelligent Systems 22(3), 48–55 (2007)

[3] Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Trans. on Knowl. and Data Eng. 17(6), 739–749 (2005)
[4] Bell, R., Koren, Y., Volinsky, C.: Modeling relationships at multiple scales to improve accuracy of large recommender systems. In: KDD 2007: Proc. of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 95–104 (2007)
[5] Deshpande, M., Karypis, G.: Item-based top-n recommendation algorithms. ACM Trans. Inf. Syst. 22(1), 143–177 (2004)
[6] Greco, G., Greco, S., Zumpano, E.: Collaborative filtering supporting web site navigation. AI Commun. 17(3), 155–166 (2004)
[7] Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. ACM Trans. Inf. Syst. 22(1), 5–53 (2004)
[8] Lee, H.J., Kim, J.W., Park, S.J.: Understanding collaborative filtering parameters for personalized recommendations in e-commerce. Electronic Commerce Research 7(3-4) (2007)
[9] Mohan, B.K., Keller, B.J., Ramakrishnan, N.: Scouts, promoters, and connectors: the roles of ratings in nearest neighbor collaborative filtering. In: EC 2006: Proc. of 7th ACM Conference on Electronic Commerce, pp. 250–259 (2006)
[10] Park, S., Pennock, D.M.: Applying collaborative filtering techniques to movie search for better ranking and browsing. In: KDD 2007: Proc. of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 550–559 (2007)
[11] Khoussainova, N., Balazinska, M., Gatterbauer, W., Kwon, Y., Suciu, D.: A case for a collaborative query management system. In: CIDR 2009: Proceedings of the 4th biennal Conference on Innovative Data Systems (2009)
[12] Singh, V., Gray, J., Thakar, A., Szalay, A.S., Raddick, J., Boroski, B., Lebedeva, S., Yanny, B.: Skyserver traffic report - the first five years. Microsoft Research, Technical Report MSR TR-2006-190 (2006)
[13] Jin, X., Zhou, Y., Mobasher, B.: Task-oriented web user modeling for recommendation. In: Ardissono, L., Brna, P., Mitrović, A. (eds.) UM 2005. LNCS, vol. 3538, pp. 109–118. Springer, Heidelberg (2005)
[14] Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. In: STOC 1996: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp. 20–29 (1996)
[15] Cormode, G., Garofalakis, M.: Sketching streams through the net: distributed approximate query tracking. In: VLDB 2005: Proceedings of the 31st international conference on Very large data bases, pp. 13–24 (2005)
[16] Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: STOC 1998: Proceedings of the thirtieth annual ACM symposium on Theory of computing, pp. 604–613 (1998)

# Scientific Mashups: Runtime-Configurable Data Product Ensembles

Bill Howe[1,⋆], Harrison Green-Fishback[2], and David Maier[2]

[1] University of Washington
billhowe@cs.washington.edu
[2] Portland State University
{hgmf,maier}@cs.pdx.edu

**Abstract.** Mashups are gaining popularity as a rapid-development, re-use-oriented programming model to replace monolithic, bottom-up application development. This programming style is attractive for the "long tail" of scientific data management applications, characterized by exploding data volumes, increasing requirements for data sharing and collaboration, but limited software engineering budgets.

We observe that scientists already routinely construct a primitive, static form of mashup—an ensemble of related visualizations that convey a specific scientific message encoded as, e.g., a Powerpoint slide. Inspired by their ubiquity, we adopt these conventional data-product ensembles as a core model, endow them with interactivity, publish them online, and allow them to be repurposed at runtime by non-programmers.

We observe that these scientific mashups must accommodate a wider audience than commerce-oriented and entertainment-oriented mashups. Collaborators, students (K12 through graduate), the public, and policy makers are all potential consumers, but each group has a different level of domain sophistication. We explore techniques for adapting one mashup for different audiences by attaching additional context, assigning defaults, and re-skinning component products.

Existing mashup frameworks (and scientific workflow systems) emphasize an expressive "boxes-and-arrows" abstraction suitable for engineering individual products but overlook requirements for organizing products into synchronized ensembles or repurposing them for different audiences.

In this paper, we articulate these requirements for scientific mashups, describe an architecture for composing mashups as interactive, reconfigurable, web-based, visualization-oriented data product ensembles, and report on an initial implementation in use at an Ocean Observatory.

## 1 Introduction

A *mashup* is a web-based, lightweight, situational application integrating services and data that were not necessarily designed to interoperate. The term was coined to describe web applications developed by the general public exercising Google's Map API, but has come to refer to any "quick and dirty" web application based on pre-existing data or services.

⋆ Work performed while author was at Oregon Health & Science University.

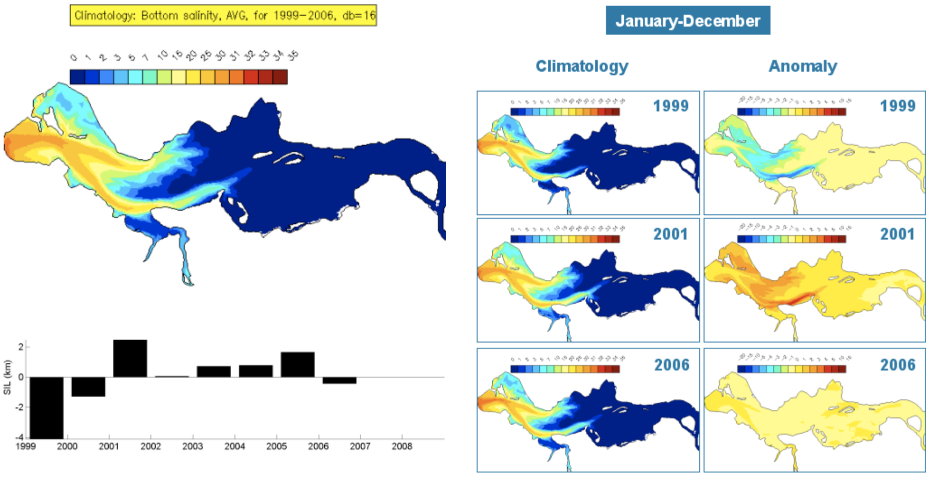


**Fig. 1.** A data product ensemble from a physical oceanography presentation on "climatologies" (long-term averages). Each map displays the Columbia River Estuary with the Pacific Ocean cropped at left and colored by bottom salinity. The large map at left is the bottom salinity averaged over the entire time period 1999-2006. The six maps at right are arranged in two colums: the left column is the yearly average, and the right column the average bottom salinity subtracted from the overall average to give an indication of variability from the mean.

The popularity of mashups is attributable to the enormous rate of collective data acquisition. Gray and Szalay argued that derived data dominates the total data volume due to pairwise comparisons [4]. That is, $N$ source datasets leads to $O(N^2)$ derived comparison datasets. Similarly, the number of mashups deployed on the web scales as $O(N^2)$ in the number of services available. For example, one application overlays crime scenes on Google's satellite images (via a join on location) [2], another links your Twitter microblog with your Flickr photstream (via a join on time) [15], another integrates your bank statements with your investment portfolio [11], and so on. The quadratic growth rate in the number of applications must be balanced by a reduction in development time, recruitment of a new class of developer, a higher degree of reuse, or all three.

Of course, personal data management and social networking applications are relatively simple problems — the amount of data processed in each operation is small (a single RSS message, a screenful of search results), and the analysis to be performed is predictable (items equipped with a latitude and longitude are displayed on a map, images are displayed in the browser using thumbnails). Many mashup frameworks rely crucially on these simplifying assumptions [6,12,20], making them inappropriate for *enterprise mashups* characterized by larger datasets, specialized users, and domain-specific processing [7]. For example, an enterprise mashup might be required to access data from relational databases, spreadsheets, documents, and other in-house proprietary data sources in order to display the combined data in a business-specific format, whereas a "consumer grade" mashup will usually access a smaller set of standard data formats, but produce result designed to be as generally accessible as possible.

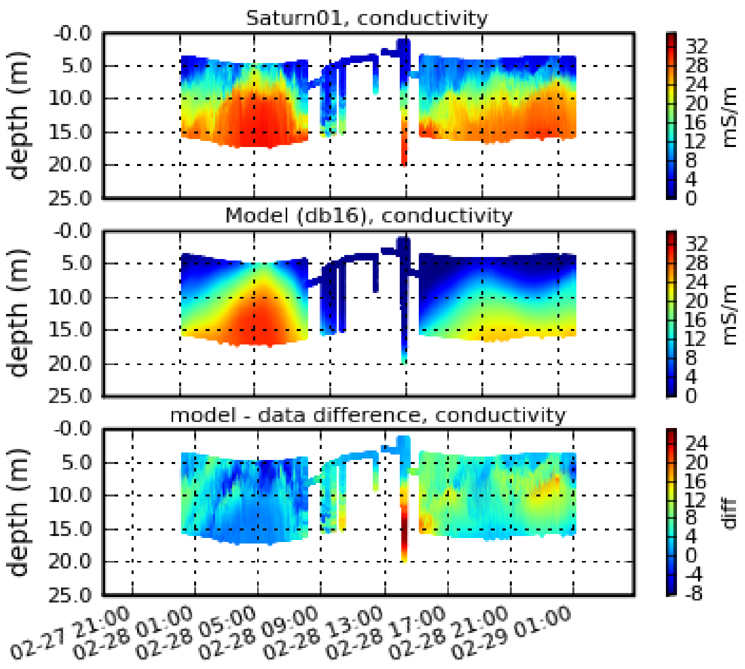


**Fig. 2.** A data product comparing observed conductivity (top), simulated conductivity (middle) and their difference (bottom) against depth from a vertically mobile platform. In our framework, this kind of static data product becomes *mashable*: parameterized, and reusable in various situational applications.

Scientific mashups push the requirements of enterprise mashups even further. In addition to large datasets and domain-specialization, scientific mashups are relevant to a much broader range of potential customers. Besides highly specialized domain experts (e.g., the seven people in the world who understand your research the best), students from K12 through post-graduate, collaborators from different fields, the press, the general public, industry colleagues, and policy makers are all potential consumers of scientific results delivered by a mashup. In contrast, an enterprise mashup is intended for a much narrower range of user types — perhaps just one or two specialized analysts or a set of identically trained customer-service agents. Further, significant scientific findings are intrinsically non-obvious, complicating their exposition. Although a map of addresses can be interpreted by nearly anyone, the meaning of Figure 1 is difficult to ascertain without explanation. (The reader is encouraged to try to interpret the ensemble before reading the explanation in Section 2.)

Scientific mashups are also expected to present rather large datasets at one time. For example, the timeseries in Figure 2 displays two days of measurements from a profiling mooring managed by the Center for Coastal Margin Observation and Prediction (CMOP). The sensor climbs up and down within the water column sampling at around 6Hz, generating over 1 million measurements over a two-day period. Apprehending the gross features of a million data points requires the higher bandwidth of the human eye — visualization must replace top-$k$ filtering and coarse statistical aggregation at this scale and complexity.

Our work aims to simplify authorship and customization of scientific, visualization-oriented "mashup" applications for diverse audiences.

## 2 Modeling Scientific Mashups

Our design of a mashup framework suitable for scientific data communication was inspired by the ubiquity of visual *data product ensembles* in scientific discourse.

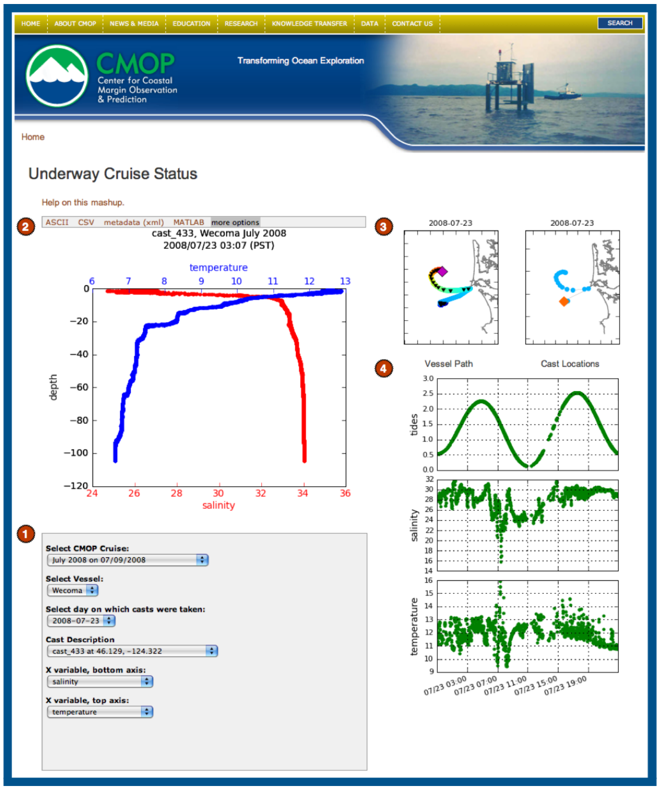
**Example:** Figure 1 provides an example of an ensemble from a Powerpoint presentation. Each map represents the Columbia River Estuary colored by the water's salinity along the river bottom as computed by the SELFE ocean-circulation model [21]. The large map at the left is the average bottom salinity from 1999-2006. The dense, salty water at the bottom is less dominated by river discharge and tidal influences, and is therefore a better characterization of estuary behavior than other variables such as temperature. The six smaller maps at the right are organized into two columns. Each row is a different year, and only data from the two-month period of December and January is shown for each year. The left column is the average bottom salinity for each of three years: 1999, 2001, 2006. The right column is the average bottom salinity subtracted from the overall average salinity, indicating anomalies. The bar chart at the lower left indicates anomolies for *salinity intrusion length* year-by-year.

This ensemble was constructed from static images by Antonio Baptista, director of CMOP, as part of a presentation on climatological (i.e., long-term) variability. Each static image was generated by the programming staff at CMOP specifically for the presentation. The ensemble was crafted to convey a specific message — that the long-term collection of *hindcast* results generated from the SELFE ocean-circulation model was accurately capturing climatological variability, including known anomalies. Specifically, this ensemble demonstrates that the model captures the fact that 1999 was fresher than average, while 2006 was unremarkable. The year 1999 is known to have exhibited extremely high river discharge in late December, so a fresh estuary agrees with observation.

### 2.1 Injecting Interactivity

The example ensemble in Figure 1 immediately suggests a family of related ensembles designed to answer related questions: Is the temperature signature anomalous in 1999 as well? Does the 1999 anomaly persist through the Spring freshet in May? This situation illustrates one goal of this research: to allow scientists to re-parameterize and re-purpose this ensemble without relying on programming staff.

To maximize the return on programmer effort, a scientific mashup framework should allow a non-programmer to reuse and repurpose programmer-crafted data products. In contrast, consider workflow systems, which also aim to raise the level of abstraction for authoring data-processing pipelines [8,16] and visual scientific applications [18,1]. Workflow systems generally support reuse — a workflow



Underway Cruise Status

Help on this mashup.

cast_433, Wecoma July 2008
2008/07/23 03:07 (PST)

Select CMOP Cruise: July 2008 on 07/09/2008

Select Vessel: Wecoma

Select day on which casts were taken: 2008-07-23

Cast Description: cast_433 at 46.129, -124.322

X variable, bottom axis: salinity

X variable, top axis: temperature

**Fig. 3.** A mashup in use at the NSF Science and Technology Center for Coastal Margin Observation and Prediction built to review profile measurements taken during research cruises. The are four main sections: 1) a chain of parameters specifying which visualizations to display, 2) the cast profile plot with two simultaneous variables on the x-axis and depth on the y-axis, 3) a pair of maps providing spatial context for the cast, and 4) a set of timeseries displaying surface measurements gathered from the vessel during the same day.

authored by one programmer can be accessed by other users of the same platform [8,16,18]. However, our stakeholders find the expressive data-flow abstractions adopted by most workflow systems to be only marginally more accessible than general purpose programming languages — and therefore effectively *inaccessible* to non-programmers.

The ensemble in Figure 1 is not designed to be interactive — it conveys a specific scientific message without requiring additional user input. In contrast, consider Figure 3. This ensemble is divided into four areas:

1. **User controls:** Users can select a completed research cruise using the topmost select widget in area (1). The vessel of interest can be chosen using the second select widget. The available vessels depend on the currently chosen cruise. The third and fourth select widgets allow the user to choose the day

the cast was taken and particular cast, respectively. The choices available in these selects are dependent on the chosen cruise and vessel. The final two select widgets allow the user to choose the two x-axis variables that will be displayed in the cast profile image.

2. **Cast profile:** The y-axis of the cast profile is depth, and the configurable top and bottom axes each reflect a measured variable.
3. **Daily Map:** The daily maps show vessel activity for the chosen day. The vessel-path map shows the vessel path colored by time of day (red is later, blue is earlier). The pink diamond shows the final vessel location at midnight. The black dots show cast locations. The cast-location map shows the cast locations as blue circles, with the selected cast highlighted in orange. These maps provide *spatial context* needed to interpret the the cast profile. For example, if a cast profile is near the estuary (as is the cast if Figure 3), then a strong freshwater signal at the surface is expected.
4. **Daily Timeseries:** The timeseries plots show tidal elevations, near-surface salinity, and near-surface temperature from the flow-through sensor package on board the vessel. The tidal elevations provide *temporal context* — the plume is fresher during low tide.

In contrast to the "display-only" mashup of Figure 1, Figure 3 involves user controls for re-parameterizing the mashup. Our mashup model erases the distinction between input and output components, allowing any *mashable* item to both display a set of values to the user, and (optionally) allow the user to select a subset of values to return to the system. Using this simple data model, we are able to express a wide variety of lightweight science applications without requiring mashup developers to learn a broad repertoire of tools. That is, every component is an instance of one underlying class: an *adapted mashable.*

## 2.2 Inferring Data Flow

Consider the following definitions. We adopt a parameterized type syntax borrowed from the polymorphic features of C++ and Java. Each type variable is a *scheme* — set of attribute names.

$$\begin{aligned}
\text{Environment}\langle E\rangle &:: E \rightarrow \text{list of strings}\\
\text{Domain}\langle T\rangle &:: \text{a relation with attributes } T\\
\text{Mashable}\langle E,T\rangle &:: \text{Environment}\langle E\rangle \rightarrow \text{Domain}\langle T\rangle\\
\text{Adaptor}\langle E,T\rangle &:: \text{Environment}\langle E\rangle \rightarrow \text{Domain}\langle T\rangle \rightarrow \text{Environment}\langle E\cup T\rangle\\
\text{AdaptedMashable}\langle E,T\rangle &:: \text{Environment}\langle E\rangle \rightarrow \text{Environment}\langle E\cup T\rangle
\end{aligned}$$

Each Mashable is a simple function abstraction for web services, database queries, shell programs, etc: given a set of parameters, return a set of tuples. Each formal parameter is a string (e.g."cruise", "date"). Parameter values are *sequences* of strings to allow multi-valued selections (e.g., a range of dates instead of a single date). Individual values are untyped; their interpretation is left up to the consumer, following conventions of REST.

Informally, each adaptor takes a set of tuples, displays them to the user, allows the user to select some subset, and then converts that subset to an environment. The keys of the resulting environment are the attributes of the source relation (as expressed in the definition of Adaptor above). For example, the first select box in area (1) of Figure 3 is populated from a relation with attributes $(cruise, startdate)$. These tuples are converted into a drop down menu by an appropriate adaptor. When the user selects a particular cruise, the adaptor (back on the server, after appropriate translations) receives the user's selection [("July 2007", "7/4/2007")]. The environment passed to the next mashable is the current environment $E$ updated with values from the new environment $T$ with keys $(cruise, startdate)$.

In this model, a *mashup* is a graph where each vertex is an AdaptedMashable (AM). Edges are derived implicitly from parameter dependency information. For example, in Figure 3, one of the select widgets allows the user to choose a particular day of the research cruise. The two maps at the upper right both make use of the selected day value to determine which data to display. No explicit link between the *day* select widget and the two context maps is required. Simply by appearing in the same scope, consumers of the *day* parameter are implicitly dependent on the producer of the *day* parameter. By synchronizing the parameters of each mashable to common sources, we reduce the chance of presenting a dangerously misleading mashup. For example, Figure 1 only make sense if all products pertain to bottom salinity — the system can help enforce this constraint by keeping all products synchronized (unless explicitly overridden by the user — see below).

**Edge Inference.** More precisely, we heuristically infer an edge between two vertices $X\langle E_X, T_X\rangle$ and $Y\langle E_Y, T_Y\rangle$ if $E_Y \subset T_X$. That is, if one AM supplies all the parameters that another AM needs, then connect them. Next, we infer edges wherever two upstream AMs together can supply the necessary parameters. That is, given a vertex $X\langle E_X, T_X\rangle$, and an edge $(Y\langle E_Y, T_Y\rangle, Z\langle E_Z, T_Z\rangle)$, infer an edge $(Z, X)$ if $E_X \subset T_Y \cup T_Z$. We continue this process for longer paths through the graph until we converge. For example, the plot in area (2) of Figure 3 (call it $P1$) requires values for *cruise*, *vessel*, and *castId*. The first select box $S1$ provides values for *cruise* and *startdate*, the second select box $S2$ requires values for *vessel* (given a *cruise*), and the third $S3$ provides values for *castid* (given a *cruise* and a *vessel*), so we infer a path in the graph $S1 \rightarrow S2 \rightarrow S3 \rightarrow P1$.

Since we rely on an implicit dependency graph among AMs, we must tolerate both *underspecified* and *overspecified* mashups. An underspecified mash-up involves AMs that require values for parameters not supplied by any other AMs. We address underspecified mashups by requiring that all mashables either adopt default values for all required parameters or tolerate their omission. For example, the right-hand map plot in area (3) of Figure 3 displays all casts for a given day, but also highlights a particular cast. The author of the underlying adaptor is expected to handle the case where no cast is specified: perhaps no cast is highlighted, or the first cast in the list is highlighted, for example. By requiring that all mashables tolerate missing parameters, we reduce the number of invalid mashups that can be expressed by the user. Anecdotally, we observe that exceptions and error messages are to be avoided at all costs: Users simply assume that the system is not working and stop using it rather than read the message and correct the problem.

An overspecified mashup provides multiple sources for the same parameter values. For example, the choice of a cast implies a temporal context: the time the cast was taken. However, the choice of a cruise also implies a temporal context: the start time of the cruise itself. A mashable that looks up the tidal elevation based on time must choose between these two timestamps, but there is not necessarily an unambiguous way to do so. In this case, we break the tie by observing that the cast is influenced by the cruise, so the cast time is in a sense more specific — it already takes into account the information supplied by the cruise. We refer to this heuristic as the *path of greatest influence* (PGI). The PGI is simply the backwards path through the directed graph that passes through largest number of competing sources for a parameter. The source to use is the nearest node along the PGI. The algorithm to implement this decision is straightforward: reverse the edges, find the longest path from the target vertex to a vertex supplying the overspecified parameter, and select the first vertex. Ties are currently broken by document order in the HTML — nearest nodes in document order are linked, under the assumption that products are usually added to the mashup in dependency order. Another tie-breaking strategy we are exploring is to automatically multiplex the overspecified product. For example, if two casts are in scope and the designer appends a tidal chart product, there is no unambiguous way to choose between them. Rather than assume the second cast is preferable to first, we can simply insert an additional copy of the tidal chart — one for each cast. This feature is not yet tested.

The implicit dependency graph is one mechanism for specifying data flow, but there are two others. Each mashup is equipped with a *root environment* configured by the mashup designer. The root environment is by default empty, but can be populated with metadata (parameter-value pairs) to resolve ambiguities in the dependency graph or to supply information that cannot be derived anywhere else. For example, the colors in Figure 1 all pertain to bottom salinity in the period 1999-2006. The pairs ($variable = bottomsalinity$, $startyear = 1999$, and $endyear = 2006$) can be inserted into the root environment. The root environment is just another node in the dependency graph — products will still pull their parameters from the nearest upstream source unless explicitly overridden by the designer.

Mashup developers may also individually set parameter values for individual products. Parameters set explicitly for individual products override other sources of parameters and allow fine-tuning of the mashup. For example, the individual years selected for the rows at the right-hand side of Figure 1 are explicitly set by the designer rather than being passed by the dataflow graph.

## 2.3 Tailoring Mashups for Specific Audiences

Consider the context provided by the static ensemble in Figure 1 for interpreting the underlying data, both explicit and implicit. The title at the upper left gives explicit context: the proper interpretation of the color (bottom salinity) and the overall time period considered (1999-2006). The fact that all the map images pertain to the same geographical region (the Columbia River Estuary) is implicit. The units of the color bar (practical salinity units, or just psu) are also implicit, since no other units are in common usage. The meaning of the y-axis label (SIL stands for Salinity Intrusion Length) is also implicit. Even knowledge of the expanded acronym does not necessarily help a novice interpret the graph. The intrusion length is in meters, but what distance is it measuring? Domain experts will understand that an estuary length is measured along its primary channel, which is usually well-defined for ship traffic, but a novice will find it difficult to develop intuition for the physics without further explanation.

We therefore seek transformations that preserve the scientific meaning of the mashup but tailor it to different audiences. We define three techniques for tailoring mashups: inserting *context products*, changing *application style*, and *re-skinning*.

The first method is to insert new products into the mashup that expose what was going on "nearby" — not just in time and space, but nearby in the overall parameter space. For example, information on tides, weather, daylight, other observational platforms, other models, and so on all potentially enhance interpretation. Additional products that simply display values in the root environment are also applicable here: the title bar in Figure 1 is an example of a simple "product."

The second method is to tailor the application style for different audiences. Anecdotally, we find that experts prefer to fill screen real estate with additional data, but novices prefer to study one product at a time to avoid feeling overwhelmed. The mashup framework supports converting from a dashboard-style interface (all products at once) to a wizard-style interface (one product at at time) without additional programming.

Finally, mashups can be re-purposed for display to different audiences by re-skinning the *mashable* components of the mashup with different *adaptors*. The mashup in Figure 3 depicts temperature as a variable on the x-axis of a cast-profile plot. This succinct approach to data visualization is desirable when the intended audience is a group of domain experts. However, this visualization is likely difficult to understand and un-engaging for elementary-school science students. In order to address this issue, the original mashup can be *re-skinned* to appeal to a wider audience. As shown in Figure 4, the product which displays depth on the y-axis and temperature and salinity on the x-axis is replaced by a product which displays depth as an animation of a cast-profiling device beneath the vessel moving up and down in the water column, with the corresponding temperature and salinity displayed as familiar graphical thermometers. In this fashion, re-skinning makes the same data product accessible to users of widely differing backgrounds and skill sets.

The discussion in this section exposes three steps in creating a scientific mashup:

1. **Wrap.** Programmers must wrap each data source as a mashable accepting an arbitrary environment of parameters mapped to sequences of values and returning a set of tuples.

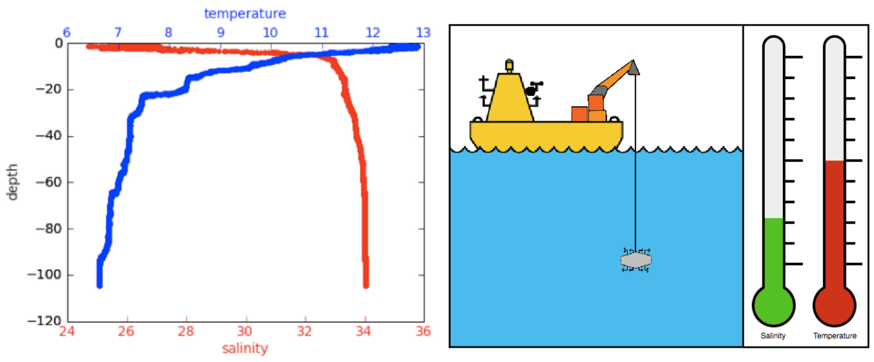


**Fig. 4.** Different adaptors can present data from the same mashable in different ways. The adaptor on the left displays depth along the y-axis and temperature and salinity along the x-axis of the cast plot. The adaptor on the right illustrates the dependency between temperature and salinity and depth by allowing the user to drag the cast probe up and down within the water column in order to see the change in temperature and salinity associated with the change in depth.

2. **Synch.** Mashup authors (usually non-programmers) choose from a set of *adapted mashables* — each one a mashable with a specific visual "skin." The set of selected mashables are wired together as a data flow graph derived using heuristics involving document order, implicit dependency information, and user input.
3. **Tailor.** Mashups can be re-purposed for different audiences by attaching additional adapted mashables, using different adaptors for the same mashables, or re-organizing into a different application style (i.e., a dense, AJAX-powered single-screen application versus a wizard-style, question-and-answer-oriented application).

## 2.4 Challenges and Limitations

The challenge of the Scientific Mashup problem is to support a broadly functional class of web applications without incurring the cognitive load associated with traditional programming. We are experimenting with an extremely simple conceptual model: mashables for retrieving data, adaptors for interacting with the user, and sequences of strings for data flow. As a result of this design choice, we rely heavily on the functionality of the mashables.

By design, we do not permit additional operations between adaptors and mashables, such as filtering, data cleaning, type conversion, or arithmetic. We assume that all such work is done inside the mashable. We impose no restrictions on the expressiveness of the mashable internals. We anticipate that some mashable components will execute complex scientific workflows to generate the domain, or otherwise execute arbitrary programs. However, because we are agnostic to the language or system used to author mashables, our work is complementary to research in programming languages, scientific workflows, and data integration systems. We are interested in empowering the non-programmer to craft interactive scientific ensembles using basic building blocks. We are exploring the question, "What are the limits to the applications can we provide assuming the user is only willing to specify 1) the visualizations they are interested in, 2) their arrangement on-screen, 3) a few parameter values in a root environment?"

Our reliance on a simplified relational model represents another limitation: data sources that return XML must be flattened into relations before they can be used with our system. We have found XML to be rather unpopular among scientific programmers for simple tasks, and we want to keep the barrier to entry for our mashable authors as low as possible.

## 3 Related Work

Workflow systems attempt to raise the level of abstraction for scientific programmers by adding language features: visual programming, provenance, limited task parallelism, fault tolerance, type-checking, sophisticated execution models [3,8,16,18]. In contrast, we adopt a top-down approach: begin with a static data product ensemble, then endow it with interactivity and publish it online.

The VisTrails system has a suite of advanced features useful as mashup support services. VisTrails exploits the graph structure of the workflow and a database of existing workflows to provide a higher-level interface for workflow composition. Users can create new workflows "by analogy" to existing workflows and the system can help "autocomplete" a workflow by offering suggestions based on the graph structure [14]. Both features rely on graph matching with an existing corpus of workflows. VisTrails also adopts a spreadsheet metaphor for displaying and browsing related visualizations. A series of workflow executions can be compared side-by-side in the cells of a grid. Further, a series of related images can be generated in one step using a parameter exploration — an iterative execution of the same workflow across a range of parameter values. Finally, the VisTrails system also allows an individual workflow to be compiled into a simple web application [19]. These features all help reduce the programmer effort required to create and re-purpose workflows. Our approach is complementary — we explore the configuration space of multiple synchronized visualizations, while remaining agnostic to how the visualizations are created.

Marini et al. describe a system to publish workflows to the web as interactive applications [9] and corroborate our emphasis on domain-specific solutions [5], but do not consider multiple synchronized workflows, nor runtime-configuration for different audiences.

There exist many commerce- and web-oriented mashup development frameworks freely available for public use [6,12,20]. From among the systems available to us we chose to examine Yahoo Pipes and Microsoft's Popfly in detail.

Yahoo Pipes is a web-based mashup solution. Pipes allows users to compose existing data feeds into new feeds that can be made available on the web. The Pipes system appears to be well suited to this task, but requires that a user be at least passingly familiar with standard Internet feed formats and XML concepts. Pipes does attempt to allow users to integrate less-structured information from arbitrary web pages into its mashups; this integration appears to be limited to displaying the contents of a page (or portion thereof) alongside structured feed information. In contrast, our requirements demand that users be able to draw from a wide range of scientific data without being forced to understand the particular storage format of that data.

Popfly is Microsoft's offering for a mashup-development framework. Popfly differentiates itself from other mashup solutions by offering a very rich user interface. Of particular interest is the feature that allows a user to examine a mashup component at different levels of abstraction. At the simplest level, a mashup component is presented as a graphical control with inputs and outputs that can be connected via drag and drop operations, similarly to Yahoo Pipes. At the next level of detail, a mashup component is presented as a choice of operations that can be performed on the inputs — integer addition, string concatenation, filtering, etc. At the lowest level of detail, a user is invited to directly edit the Javascript code behind a mashup component. In this fashion, different designers with varying levels of programming expertise can interact with Popfly in the fashion that best suits their needs and abilities — all within the same development interface. This ability to drill down to a preferred level of programming-interface complexity is well-suited to a scientific mashup application where a substantial number of users can be expected to have advanced mathematical skills as well as a reasonably advanced programing skill set. However, data processing in Popfly is performed in client-side Javascript, which is inefficient in the context of the large data sets and computationally expensive operations inherent in a scientific mashup.

We observe that software vendors are increasingly aware of audience-adaptibility. Microsoft Office products hide unused options from their menus in order to avoid overwhelming novice users. Tax preparation applications (e.g. TurboTax [17]) provide multiple application styles for the same core content. A questionnaire application style (called EasyStep in TurboTax) can be used in conjunction with a form-oriented application style designed for experts. We are studying how these techniques may be adopted for scientific mashups.

## 4 System Architecture

Our initial mashup system is comprised of five major components. The environment, the mashable, the domain, the adaptor, and the mashup engine.

1. **Environment.** An environment is a map of keys to values and is the input to a mashable. All mashups are designed to accommodate an empty or default environment. Thus a valid product ensemble is guaranteed even in the absence of user interaction.
2. **Mashable.** The mashable is the primary means of data-source abstraction. A mashable takes an environment as input and produces a domain as output. Mashables are reusable components created by programmers for end users to work with during runtime configuration of a mashup.
3. **Domain.** A domain is a set of tuples returned by a mashable.
4. **Adaptor.** Adaptors render domains. All user-interface components are adaptors. Adaptors can be display-only or interactive. An interactive adaptor

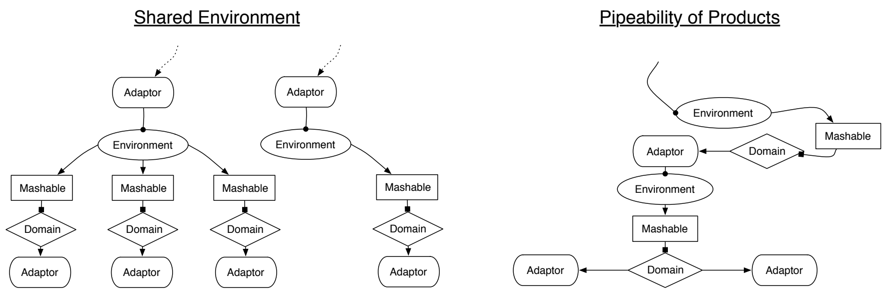


**Fig. 5.** Possible mashup graph configurations

allows a user to modify the environment passed to subsequent mashables in the mashup graph. Like mashables, adaptors are created by programmers for configuration within a mashup by end users. Different adaptors may be associated with a single mashable, allowing audience-specific presentations of data.

5. **Mashup Engine.** The mashup engine is the context within which the other mashup components exist. Designers browse for mashables and adaptors that have been defined previously and link them together at runtime within the mashup engine. The mashup engine is responsible for automatically synchronizing the mashup components when a user alters the state of an environment via interaction with an adaptor.

It is possible to have multiple products share a single environment, as illustrated in Figure 5 (left). Constructing a mashup in this fashion allows user interaction with a single Adaptor to effect the state of all products that are members of the mashup tree rooted at the shared environment. Products located within a separate subtree of the mashup are not affected by this adaptor interaction. A users interaction with a given product within a product ensemble affects the state of the environment. A cloned instance of the environment is passed down from product to product as in Figure 5 (right). In this way the result of interaction with one product in the chain is passed down to subsequent products in the product chain.

Listing 1.1 provides an example of the mechanisms used to wire a set of mashables and adaptors together in order to create a functioning mashup. In this example, the mashup wiring is shown explicitly. However, similar wiring might just as easily take place at runtime as the result of a user's interaction with the system.

In line 1, a new instance of a mashup is created. The name (CmopTest) of the mashup serves to identify this particular mashup within a system that may host many different mashup instances at the same time. In lines 3 and 4, a Mashable reference to the CmopVesselMashable is obtained and a new SelectAdaptor is created and named. The CmopVesselMashable is an implementation of Mashable that abstracts vessel-specific data within the CMOP database. The SelectAdaptor is an adaptor implementation that renders its adapted domain as an HTML select widget. In lines 5 and 6, we define a mapping to associate the attributes

**Listing 1.1.** A mashup example linking two select widgets

```
1  Mashup m = new MashupImpl(“CmopTest”);

3  Mashable vessel = CmopVesselMashable.getInstance();
4  Adaptor selectvessel = new SelectAdaptor(“vessel”);
5  Map<String,String> vesselmap = new HashMap<String,String>();
6  vesselmap.put(SelectAdaptor.KEY, “vessel”);
7  m.addMashable(vessel);
8  m.linkAdaptor(selectvessel, vessel, vesselmap);

10 Mashable cruise = CmopCruiseMashable.getInstance();
11 Adaptor selectcruise = new SelectAdaptor(“cruise”);
12 Map<String,String> cruisemap = new HashMap<String,String>();
13 cruisemap.put(SelectAdaptor.KEY, “cruise”);
14 m.addMashable(cruise);
15 m.linkAdaptor(selectcruise, cruise, cruisemap);

17 m.linkMashable(cruise, vessel);
```

required by the adaptor with attributes available within the domain produced by the adapted mashable. In lines 7 and 8, the mashable instance is added to the mashup and linked to the configured adaptor.

In lines 10-15, a new mashable and adaptor are linked together and added to the mashup. In this case, the mashable is an abstraction of CMOP cruise data and the adaptor is again an instance which renders as an html select widget.

Finally, in line 17, the two mashables are linked or “mashed” together within the context of the mashup. In this case the cruiseMashable is the “masher” and the vesselMashable is the “mashee”. As such, changes to the cruiseSelectAdaptor made by the user will modify the environment provided to the vesselMashable. This changed environment will cause a corresponding change to the domain produced by the vesselMashable and ultimately to the data displayed to the user by the vesselSelectAdaptor.

## 5 A Mashup Factory for an Ocean Observatory

In Section 4, we described an initial implementation of the complete mashup model. In this section, we describe an earlier incarnation of these ideas called the *Product Factory*. The Product Factory is currently deployed at the Center for Coastal Margin Observation and Prediction, and is part of the infrastructure for the upcoming Pacific FishTrax website sponsored by the project for Collaborative Research on Oregon Ocean Salmon (CROOS) [13]. The design goals of the Product Factory are:

1. Replace static images with dynamic, user-configurable data products.
   **Method:** *Distill each product to a set of parameters, an SQL statement, and a short plotting script. All other boilerplate code is provided by the factory.*
2. Simplify the creation of web applications involving multiple data products.
   **Method:** *Expose each factory product as a RESTful web service, allowing*

*data products, parameter values, and source data to be embedded in HTML without additional server-side programming.*

3. Provide public access to the underlying data, not just the visualization.
   **Method:** *Registering a product in the factory automatically establishes a web service for data access in a variety of common formats.*

To create a new product, a programmer writes a *product specification* in either XML or Python. An example XML product specification appears in Listing 1.2. The specification involves three sections: a set of parameters (lines 3-15), an *extractwith* clause indicating how to extract data (lines 16-21), and a *plotwith* clause indicating how to render the data (lines 22-28).

The Product Factory extracts data and parameter domains from a relational database. Generalization of the Factory to allow access to arbitrary data sources was an important motivation in designing the architecture of Section 4.

Each parameters has a type that determines its behavior. The base type `Parameter` allows unconstrained user input, and is rendered as a simple text box. A `SelectParameter` (lines 13-15) takes a comma-delimited list of strings and forces the use to select one using an HTML select tag. A `SQLSelectParameter` (e.g., lines 3-5) is similar to a `SelectParameter`, but the choices are drawn from a database using an SQL statement. SQL-powered parameters may depend on the values of earlier parameters. Syntactically, the SQL statement may include embedded placeholders using Python string-formatting conventions. In the HTML interface, we use asynchronous Javascript to dynamically refresh the values of downstream parameters when upstream parameters change.

Other parameter types available include `MultiSelect` and `SQLMultiSelect` versions that allow multiple option to be selected by the user, `HiddenParameter` that computes a value for downstream processing but does not interact with the user, and `DynamicDefaultParameter` that computes an initial value from upstream parameters but allows arbitrary user input. Each SQL-driven parameters is similar to the product itself — data is extracted from the database and displayed to the user. The observation that parameters are not fundamentally different from products led to their unification in the current model (Section 2).

Like the SQL-powered parameters, the `extractwith` clause uses string-substitution placeholders to reference parameter values. The `plotwith` clause is simply Python code executed in an environment with all parameters and plotting libraries pre-loaded as local variables. Most `plotwith` clauses are remarkably short, using just a few MATLAB-style calls provided by the matplotlib 2D plotting library [10]. Since many of our programmers are familiar with MATLAB, matplotlib provides an attractive alternative for programming visualizations.

Each parameter type specifies how to compute a default value. For the base `Parameter` type, the default value is given explicitly in the definition. Parameters with explicit domains use the first value in the domain as the default value. If a parameter's domain is empty (e.g., the SQL query returns no records), then a sentinel value akin to NULL is returned. A downstream parameter may or may not be designed to tolerate NULL — if an exception is raised, the downstream parameter is itself assigned NULL. This aggressive propagation of NULL values is necessary to prevent exception messages and errant behavior for the end user

**Listing 1.2.** A factory product specification in XML. The Product Factory is an early implementation of a scientific mashup framework focused on reducing the code required to publish an interactive data product.

```
<specification>
 <product name=``castprofile">
  <parameter name=``vessel" type=``SQLSelectParameter">
    SELECT vessel FROM cruise.vessel
  </parameter>
  <parameter name=``cruise" type=``SQLSelectParameter">
    SELECT cruise FROM cruise.cruise WHERE vessel='%(vessel)s'
  </parameter>
  <parameter name=``cast" type=``SQLSelectParameter">
    SELECT distinct castid, castdescription FROM ctdcast
     WHERE vessel = '%(vessel)s' AND cruise = '%(cruise)s'
  </parameter>
  <parameter name="variable" type="SelectParameter">
   salinity, conductivity, temperature, pressure, turbidity
  </parameter>
  <extractwith>
    SELECT time, -depth as depth, %(variable)s as variabledata
      FROM castobservation
     WHERE vessel = '%(vessel)s'
       AND cruise = '%(cruise)s' AND castid = '%(cast)s'
  </extractwith>
  <plotwith>
  title("%s %s %s' % (cast,vessel,cruise), size='small')
  scatter(variabledata, depth, faceted=False)
  xlabel(variable, size='small')
  ylabel('depth (m)', size='small')
  </plotwith>
 </product>
</specification>
```

— in our experience, a product that returns an empty dataset is tolerable, but error messages are not. However, masking exceptions complicates debugging, so we provide a command line tool for testing products in a controlled environment.

The Product Factory simplifies the Wrap step of mashup authorship by changing the skillsets required to publish interactive data products on the web. Instead of learning a host of languages and configuration tools, programmers can simply write a series of related SQL statements and a short MATLAB-style script — higher-level language skills that are generally easier to learn (and that CMOP programmers happen to already possess!).

To register a specification with the Factory, the XML script is uploaded through either a command-line tool or a web form. The script is first tested in the empty environment (all products must produce meaningful results with no user decisions), then loaded into the factory database. Once loaded, rendered products, source data, parameter domains, and other information are all available through a RESTful web service interface. Factory calls are of the form

```
http://server.com?request=<r>&product=<prd>&<p1>=<v1>&...
```

where `r` is one of `getproduct`, `getdata`, `getdomain`, `getspec`, `prd` is the name of a product, `pi` is a parameter name and `vi` is a url-encoded value.

To construct a synchronized ensemble from individual products as in Figure 3, a mashup designer need only include each relevant product in the HTML page using the syntax `<div class="factory" product="cast"/>` and include the factory Javascript library. Each product will be expanded into either an HTML form or a simple image tag, depending on whether or not its parameters need to be displayed. For example, in Figure 3, areas (1) and (2) are together an HTML form rendered for the cast profile product (similar but not identical to Listing 1.2). Any other products that share these parameters can "borrow" from the castprofile product, thereby avoiding the need to have the user specify the relevant cruise, vessel, and cast multiple times. This mechanism supports the Synch step of mashup authorship — product "building blocks" can be brought into the same scope, and the system will derive a wiring diagram for them based on the dependency graph between parameters and products.

## 6 Conclusions and Future Work

The adoption of a mashup style of application development is potentially transformative for scientific communication. Faced with exploding data volumes, enormous data heterogeneity, and limited programming staff, development of new data product ensembles is becoming the bottleneck to dissemination of results. Our model unifies visualizations with interactive user controls, providing a simple underlying model that can express a wide variety of scientific applications. The ability to adapt existing mashups to tailor them for new audiences provides another dimension of reuse. The initial deployment of the Product Factory provides evidence that a successful mashup framework should not just raise the level of abstraction for individual data products, but also provide tools for organizing constituent products into interactive data ensembles. With this approach, scientists are empowered to reuse programmers' work (via support for product synchronization), and the end users are empowered to reuse a scientists' work (via interactive user controls).

Future work includes a formalization of the context model outlined in Section 2. We hope to be able to quantify interpretability based on domain-specific models of context. Equipping the mashup framework with semantics may allow a broader and more useful consideration of context and interpretability. Endowing individual mashable components with semantics will also allow more advanced reasoning by the system. The current implementation relies too heavily on advanced programming skillsets. Mashup designers must at least write HTML to connect mashable components together, but we plan a drag-and-drop interface similar to Microsoft's Popfly or Yahoo Pipes.

## Acknowledgements

We wish to thank Antonio Baptista, Roger Barga, Juliana Friere, and Emanuele Santos for helpful discussions. This work was supported by funding from the NSF

STC for Coastal Margin Observation and Prediction, a Jim Gray Seed Grant from Microsoft, and the eScience Institute at the University of Washington.

## References

[1] Barga, R., Jackson, J., Araujo, N., Guo, D., Gautam, N., Grochow, K., Lazowska, E.: Trident: Scientific Workflow Workbench for Oceanography. In: IEEE Congress on Services, pp. 465–466. IEEE Computer Society, Los Alamitos (2008)
[2] CrimeMapping (2008), `http://www.crimemapping.com/`
[3] Deelman, E., Singh, G., Su, M.H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A., Jacob, J.C., Katz, D.S.: Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. Scientific Programming Journal 13(3), 219–237 (2005)
[4] Gray, J., Szalay, A.S.: Where the Rubber Meets the Sky: Bridging the Gap between Databases and Science. IEEE Data Eng. Bull. 27(4), 3–11 (2004)
[5] Hill, D.J., Minsker, B., Liu, Y., Myers, J.: End-to-End Cyberinfrastructure for Real-Time Environmental Decision Support. In: IEEE eScience (2008)
[6] JackBe, `http://www.jackbe.com`
[7] Jhingran, A.: Enterprise Information Mashups: Integrating Information, Simply. In: 32nd International Conference on Very Large Data Bases (2006)
[8] The Kepler Project, `http://kepler-project.org`
[9] Marini, L., Kooper, R., Bajcsy, P., Myers, J.D.: Publishing Active Workflows to Problem-Focused Web Spaces. In: IEEE eScience (2008)
[10] The Matplotlib Library, `http://matplotlib.sourceforge.net`
[11] Mint.com, `http://mint.com`
[12] Microsoft Popfly, `http://www.popfly.com`
[13] Collaborative Research on Oregon Ocean Salmon project (ProjectCROOS), `http://projectcroos.com`
[14] Silva, C.: VisTrail, personal communication (2008)
[15] SnapTweet, `http://snaptweet.com/`
[16] The Taverna Project, `http://taverna.sourceforge.net`
[17] TurboTax, `http://turbotax.intuit.com`
[18] The VisTrails Project, `http://www.vistrails.org`
[19] Santos, E., Freire, J., Silva, C.: Using Workflow Medleys to Streamline Exploratory Tasks, `http://www.research.ibm.com/gvss/2007/presentations/emanuele_ibm_gvss2007.pdf`
[20] Yahoo Pipes, `http://pipes.yahoo.com/pipes`
[21] Zhang, Y.L., Baptista, A.M.: SELFE: A semi-implicit Eulerian-Lagrangian finite-element model for cross-scale ocean circulation. Ocean Modelling 21(3-4), 71–96 (2008)

# View Discovery in OLAP Databases through Statistical Combinatorial Optimization

Cliff Joslyn[1], John Burke[1], Terence Critchlow[1], Nick Hengartner[2], and Emilie Hogan[1,3]

[1] Pacific Northwest National Laboratory
[2] Los Alamos National Laboratory
[3] Mathematics Department, Rutgers University

**Abstract.** The capability of OLAP database software systems to handle data complexity comes at a high price for analysts, presenting them a combinatorially vast space of views of a relational database. We respond to the need to deploy technologies sufficient to allow users to guide themselves to areas of local structure by casting the space of "views" of an OLAP database as a combinatorial object of all projections and subsets, and "view discovery" as an search process over that lattice. We equip the view lattice with statistical information theoretical measures sufficient to support a combinatorial optimization process. We outline "hop-chaining" as a particular view discovery algorithm over this object, wherein users are guided across a permutation of the dimensions by searching for successive two-dimensional views, pushing seen dimensions into an increasingly large background filter in a "spiraling" search process. We illustrate this work in the context of data cubes recording summary statistics for radiation portal monitors at US ports.

## 1 Introduction and Related Work

OnLine Analytical Processing (OLAP) [6,7] is a relational database technology providing users with rapid access to summary, aggregated views of a single large database, and is widely recognized for knowledge representation and discovery in high-dimensional relational databases. OLAP technologies provide intuitive and graphical access to the massively complex set of possible summary views available in large relational (SQL) structured data repositories [21]. But the ability of OLAP database software systems, such as the industry-leading Hyperion[1] and ProClarity[2] platforms, to handle data complexity comes at a high price for analysts. The available portions and projections of the overall data space present a bewilderingly wide-ranging, combinatorially vast, space of options. There is an urgent need for knowledge discovery techniques that guide users' knowledge

[1] `http://www.oracle.com/technology/products/bi/essbase/visual-explorer.html`

[2] `http://www.microsoft.com/bi/products/ProClarity/proclarity-overview.aspx`

discovery tasks; to find relevant patterns, trends, and anomalies; and to do so within the intuitive interfaces provided by “business intelligence” OLAP tools.

For example, consider a decision-maker responsible for analyzing a large relational database of records of events of personal vehicles, cargo vehicles, and others passing through radiation portal monitors (RPMs) at US ports of entry. In our data cubes include dimensions for multiple time representations, spatial hierarchies of collections of RPMs at different locations, and RPM attributes such as vendor. In this context, a vast collection of different views, focusing on different combinations of dimensions, and different subsets of records, are available to the user. How can the user assess the relative significance of different views? Given, for example, an initial view focusing on RPM type and date, is it more significant to focus on passenger or cargo RPMs, or a particular month in the year? And given such a selection, is it more significant to next consider a spatial dimension, or any of more than a dozen independent dimensions available?

Through the Generalized Data-Driven Analysis and Integration (GDDAI) Project [11], our team has been developing both pure and hybrid OLAP data analysis capabilities for a range of homeland security applications. The overall GDDAI goal is to provide a seamless integration of analysis capabilities, allowing analysts to focus on understanding the *data* instead of the *tools*. We describe GDDAI’s approach to knowledge discovery in OLAP data cubes using information-theoretical combinatorial optimization, and as applied in the ProClarity platform on databases of surveillance data from radiation monitors at US ports of entry. We aim at a formalism for user-assisted knowledge discovery in OLAP databases around the fundamental concept of **view chaining**. Users are provided with analytical feedback to guide themselves to areas of high local structure within view space: that is, to significant collections of dimensions and data items (columns and rows, respectively), in an OLAP-structured database.

OLAP is fundamentally concerned with a collection of $N$ variables $X^i$ and a multi-dimensional data relation over their Cartesian product $\mathbf{X} := \bigtimes_{i=1}^{N} X^i$. Thus formalisms for OLAP data analysis are naturally rooted in relational database theory, and OLAP formalisms [1,10,28] extend relational calculi and algebras, for example extending the SQL language to its multi-dimensional analog MDX[3]. But OLAP shares mathematical connections with a range of multivariate analytical approaches operable on the space $\mathbf{X}$, for example statistical databases [26]; the analysis of contingency tables [3]; hierarchical log-linear modeling [18]; grand tour methods in multi-variate data visualization [2]; projection pursuit [19]; data tensor analysis [14]; and reconstructibility analysis [13,20].

Our ultimate goal is to place OLAP knowledge discovery methods within a mathematical context of combinatorial optimization in such a manner as to be realizable within existing industry-standard database patforms. Specifically:

[3] `http://msdn.microsoft.com/en-us/library/ms145506.aspx`

- Given a foundational OLAP database engine platform (e.g. EssBase[4], SSAS[5]);
- And an OLAP client with technology for graphical display and an intuitive interface (e.g. ProClarity, Hyperion);
- Cast the space of **views** (sub-cubes) of an OLAP database as a combinatorial, lattice-theoretical [9] structure;
- Equipped with statistical measures reflecting the structural relations among views (their dimensional scope, depth, etc.) in the context of the data observed within them;
- To support both automated search to areas of high local structure;
- And user-guided exploration of views in the context of these measures.

While we believe that our emphasis on a combinatorial approach is distinct, our work resonates with that of a number of others. Our "view chaining" (moving from one projected subset of a data cube to another intersecting in dimensionality) is similar to the navigational processes described by others [23,24,25], and anticipated in some of our prior work [12]. But approaches which seek out "drill-down paths" [5] only "descend" the view lattice along one "axis" of views with increasing dimensional extension, sequentially adding variables to the view at each step. In contrast, our "hop-chaining" technique chains through a sequence of two-dimensional views, affecting a permutation of the variables $X^i$.

Our overall approach is consistent with an increasingly large body of similar work drawing on information theoretical statistical measures in data cubes to provide quantities for making navigational choices [20,22]. However, some other researchers have used different statistical approaches, for example variance estimation [25] or skewness measures [16]. Our primary departure from traditional OLAP analysis is the extension to conditioning and conditional probability measures over views. This not only provides the basis for optimization and navigation, it also creates a strong connection to graphical or structural statistical models [4,27], graphoid logics [17,27], as well as systems-theory based structural model induction methodologies [13,15].

We begin by establishing concepts and notation for (non-hierarchical) OLAP databases over data tensors, and then define the **view lattice** of projected subsets over such structures. This brings us to a point where we can explicate the fundamental (again non-hierarchical) OLAP operations of projection, extension, filtering, and "flushing" (decreasing a filter). We introduce **conditional views** and the complex combinatorial object which is the **conditional view space**. This prepares us to introduce "hop-chaining" as a particular view discovery algorithm over this combinatorial object, wherein users are guided by conditional information measures across a permutation of the dimensions by searching for successive two-dimensional views, pushing seen dimensions in a "spiraling" search process into an increasingly large background filter. We then consider how to move to the fully hierarchical case, before illustrating hop-chaining on databases of surveillance data from radiation monitors at US ports of entry.

[4] `http://www.oracle.com/appserver/business-intelligence/essbase.html`
[5] `http://msdn.microsoft.com/en-us/library/ms175609(SQL.90).aspx`

## 2 OLAP Formalism

Although the mathematical tools required to analyze OLAP databases are relatively simple, their notational formalisms are inevitably, and regretably, not [1,10,28]. While our formalism is similar, it differs in a number of ways as well:

- We combine projections $I$ on dimensions and restrictions $J$ on records into a lattice-theoretical object called a **view** $\mathcal{D}_{I,J}$.
- OLAP concerns databases organized around collections of variables which can be distinguished as: dimensions, which have a hierarchical structure, and whose Cartesian product forms the data cube's schema; and measures, which can be numerically aggregated within different slices of that schema. For this work we consider cubes with a single integral measure, which in our application is the count of a number of records in the underlying database. While in principle any numerical measure could yield, through appropriate normalization, frequency distributions for use in our view discovery technique, these count measures do so directly and naturally. In future work we will consider the generalization to arbitrary numerical measures.
- Our view discovery method is currently only available on flat dimensions which are not hierarchically-structured to support roll-up aggregation and drill-down disaggregation operations. Future directions to extend to fully hierarchical OLAP data cubes will be indicated in Sec. 5.3.

### 2.1 Chaining Operations in the View Lattice of Data Tensor Cubes

Let $\mathbb{N} = \{1, 2, \ldots\}, \mathbb{N}_N := \{1, 2, \ldots, N\}$. For some $N \in \mathbb{N}$, define a **data cube** as an $N$-dimensional tensor $\mathcal{D} := \langle \mathbf{X}, \mathcal{X}, c \rangle$ where:

- $\mathcal{X} := \{X^i\}_{i=1}^N$ is a collection of $N$ **variables** or **columns** with $X^i := \{x_{k^i}\}_{k^i=1}^{L^i} \in \mathcal{X}$;
- $\mathbf{X} := \bigtimes_{X^i \in \mathcal{X}} X^i$ is a **data space** or **data schema** whose members are $N$-dimensional vectors $\boldsymbol{x} = \langle x_{k^1}, x_{k^2}, \ldots, x_{k^N} \rangle = \langle x_{k^i} \rangle_{i=1}^N \in \mathbf{X}$ called **slots**;
- $c : \mathbf{X} \to \{0, 1, \ldots\}$ is a **count** function.

Let $M := \sum_{\boldsymbol{x} \in \mathbf{X}} c(\boldsymbol{x})$ be the total number of records in the database. Then $\mathcal{D}$ also has relative frequencies $f$ on the cells, so that $f : \mathbf{X} \to [0, 1]$, where $f(\boldsymbol{x}) = \frac{c(\boldsymbol{x})}{M}$, and thus $\sum_{\boldsymbol{x} \in \mathbf{X}} f(\boldsymbol{x}) = 1$. An example of a data tensor with simulated data for our RPM cube is shown in Table 1, for $\mathcal{X} = \{X^1, X^2, X^3\} =$ { RPM Manufacturer, Location, Month }, with RPM Mfr = { Ludlum, SAIC }, Location = { New York, Seattle, Miami }, and Month = { Jan, Feb, Mar, Apr }, so that $N = 3$. The table shows the counts $c(\boldsymbol{x})$, so that $M = 74$, and the frequencies $f(\boldsymbol{x})$.

At any time, we may look at a projection of $\mathcal{D}$ along a sub-cross-product involving only certain dimensions with indices $I \subset \mathbb{N}_N$. Call $I$ a **projector**, and denote $\boldsymbol{x} \downarrow I = \langle x_{k^i} \rangle_{i \in I} \in \mathbf{X} \downarrow I$ where $\mathbf{X} \downarrow I := \bigtimes_{i \in I} X^i$, as a projected vector

**Table 1.** An example data tensor. Blank entries repeat the elements above, and rows with zero counts are suppressed.

| RPM Mfr | Location | Month | $c(\boldsymbol{x})$ | $f(\boldsymbol{x})$ |
|---|---|---|---|---|
| Ludlum | New York | Jan | 1 | 0.014 |
| | | Mar | 3 | 0.041 |
| | | Apr | 7 | 0.095 |
| | Seattle | Jan | 9 | 0.122 |
| | | Apr | 15 | 0.203 |
| | Miami | Jan | 2 | 0.027 |
| | | Feb | 8 | 0.108 |
| | | Mar | 4 | 0.054 |
| | | Apr | 1 | 0.014 |

| RPM Mfr | Location | Month | $c(\boldsymbol{x})$ | $f(\boldsymbol{x})$ |
|---|---|---|---|---|
| SAIC | New York | Jan | 1 | 0.014 |
| | Seattle | Feb | 4 | 0.054 |
| | | Mar | 3 | 0.041 |
| | | Apr | 3 | 0.041 |
| | Miami | Jan | 6 | 0.081 |
| | | Feb | 2 | 0.027 |
| | | Mar | 4 | 0.054 |
| | | Apr | 1 | 0.014 |

and data schema. We write $\boldsymbol{x} \downarrow i$ for $\boldsymbol{x} \downarrow \{i\}$, and for projectors $I \subseteq I'$ and vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbf{X}$, we use $\boldsymbol{x} \downarrow I \subseteq \boldsymbol{y} \downarrow I'$ to mean $\forall i \in I, \boldsymbol{x} \downarrow i = \boldsymbol{y} \downarrow i$.

Count and frequency functions convey to the projected count and frequency functions denoted $c[I] : \mathbf{X} \downarrow I \to \mathbb{N}$ and $f[I] : \mathbf{X} \downarrow I \to [0, 1]$, so that

$$c[I](\boldsymbol{x} \downarrow I) = \sum_{\boldsymbol{x}' \downarrow \mathbb{N}_N \supseteq \boldsymbol{x} \downarrow I} c(\boldsymbol{x}') \tag{1}$$

$$f[I](\boldsymbol{x} \downarrow I) = \sum_{\boldsymbol{x}' \downarrow \mathbb{N}_N \supseteq \boldsymbol{x} \downarrow I} f(\boldsymbol{x}'), \tag{2}$$

and $\sum_{\boldsymbol{x} \downarrow I \in \mathbf{X} \downarrow I} f[I](\boldsymbol{x} \downarrow I) = 1$. In words, we add the counts (resp. frequencies) over all vectors in $\boldsymbol{y} \in \mathbf{X}$ such that $\boldsymbol{y} \downarrow I = \boldsymbol{x} \downarrow I$. This is just the process of building the $I$-marginal over $f$, seen as a joint distribution over the $X^i$ for $i \in I$.

Any set of record indices $J \subseteq \mathbb{N}_M$ is called a **filter**. Then we can consider the filtered count function $c^J : \mathbf{X} \to \{0, 1, \ldots\}$ and frequency function $f^J : \mathbf{X} \to [0, 1]$ whose values are reduced by the restriction in $J \subseteq \mathbb{N}_M$, now determining

$$M' := \sum_{\boldsymbol{x} \in \mathbf{X}} c^J(\boldsymbol{x}) = |J| \leq M. \tag{3}$$

We renormalize the frequencies $f^J$ over the resulting $M'$ to derive

$$f^J(\boldsymbol{x}) = \frac{c^J(\boldsymbol{x})}{M'}, \tag{4}$$

so that still $\sum_{\boldsymbol{x} \in \mathbf{X}} f^J(\boldsymbol{x}) = 1$.

Finally, when both a selector $I$ and filter $J$ are available, then we have $c^J[I] : \mathbf{X} \downarrow I \to \{0, 1, \ldots\}, f^J[I] : \mathbf{x} \downarrow I \to [0, 1]$ defined analogously, where now $\sum_{\boldsymbol{x} \downarrow I \in \mathbf{X} \downarrow I} f^J[I](\boldsymbol{x} \downarrow I) = 1$. So given a data cube $\mathcal{D}$, denote $\mathcal{D}_{I,J}$ as a **view** of $\mathcal{D}$, restricting our attention to just the $J$ records projected onto just the $I$ dimensions $\mathbf{X} \downarrow I$, and determining counts $c^J[I]$ and frequencies $f^J[I]$.

In a lattice theoretical context [9], each projector $I \subseteq \mathbb{N}_N$ can be cast as a point in the Boolean lattice $\mathcal{B}^N$ of dimension $N$ called a **projector lattice**. Similarly, each filter $J \subseteq \mathbb{N}_M$ is a point in a Boolean lattice $\mathcal{B}^M$ called a **filter**

**lattice**. Thus each view $\mathcal{D}_{I,J}$ maps to a unique node in the **view lattice** $\mathcal{B} := \mathcal{B}^N \times \mathcal{B}^M = 2^N \times 2^M$, the Cartesian product of the projector and filter lattices.

We then define **chaining** operations as transitions from an initial view $\mathcal{D}_{I,J}$ to another $\mathcal{D}_{I',J}$ or $\mathcal{D}_{I,J'}$, corresponding to a move in the view lattice $\mathcal{B}$:

**Projection:** Removal of a dimension so that $I' = I \setminus \{i\}$ for some $i \in I$. This corresponds to moving a single step down in $\mathcal{B}^N$, and to marginalization in statistical analyses. We have $\forall \boldsymbol{x}' \downarrow I' \in \mathbf{X} \downarrow I'$,

$$c^J\,[I']\,(\boldsymbol{x}' \downarrow I') = \sum_{\boldsymbol{x}\downarrow I \supseteq \boldsymbol{x}'\downarrow I'} c^J[I](\boldsymbol{x}). \tag{5}$$

**Extension:** Addition of a dimension so that $I' = I \cup \{i\}$ for some $i \notin I$. This corresponds to moving a single step up in $\mathcal{B}^N$. We're now *disaggregating* or *distributing* information about the $I$ dimensions over the $I' \setminus I$ dimensions. Notationally, we have the converse of (5), so that $\forall \boldsymbol{x} \downarrow I \in \mathbf{X} \downarrow I$,

$$\sum_{\boldsymbol{x}'\downarrow I' \supseteq \boldsymbol{x}\downarrow I} c^J[I'](\boldsymbol{x}') = c^J\,[I]\,(\boldsymbol{x} \downarrow I).$$

**Filtering:** Removal of records by strengthening the filter, so that $J' \subseteq J$. This corresponds to moving potentially multiple steps down in $\mathcal{B}^M$.

**Flushing:** Addition of records by weakening (reversing, flushing) the filter, so that $J' \supseteq J$. Corresponds to moving potentially multiple steps up in $\mathcal{B}^M$.

Repeated chaining operations thus map to trajectories in $\mathcal{B}$. Consider the very small example shown in Fig. 1 for $N = M = 2$ with dimensions $\mathcal{X} = \{X, Y\}$ and two $N$-dimensional data vectors $\boldsymbol{a}, \boldsymbol{b} \in X \times Y$, and denote e.g. $X/\boldsymbol{ab} = \{\boldsymbol{a} \downarrow \{X\}, \boldsymbol{b} \downarrow \{X\}\}$. The left side of Fig. 1 shows the separate projector and selector lattices (bottom nodes $\emptyset$ not shown ), with extension as a transition to a higher rank in the lattice and projection as a downward transition. Similarly, filtering and flushing are the corresponding operations in the filter lattice. The view lattice is shown on the right, along with a particular chain operation $\mathcal{D}_{\{X,Y\},\{\boldsymbol{a}\}} \mapsto \mathcal{D}_{\{X\},\{\boldsymbol{a}\}}$, which projects the subset of records $\{\boldsymbol{a}\}$ from the two-dimensional view $\{X, Y\} = \mathcal{X}$ to the one-dimensional view $\{X\} \subseteq \mathcal{X}$.

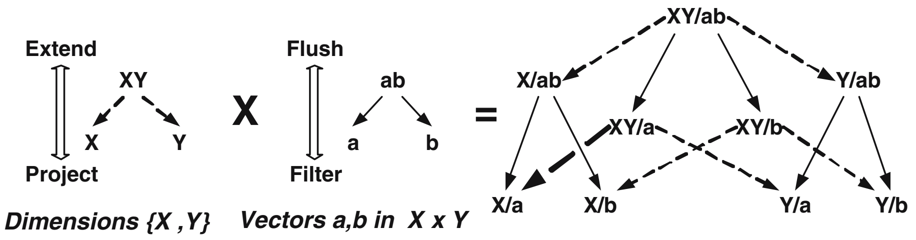


**Fig. 1.** he lattice theoretical view of data views. (Left) The projector and filter lattices $\mathcal{B}^N, \mathcal{B}^M$ (global lower bounds $\emptyset$ not shown). (Right) The view lattice $\mathcal{B}$ as their product. The projection chain operation $\mathcal{D}_{\{X,Y\},\{\boldsymbol{a}\}} \mapsto \mathcal{D}_{\{X\},\{\boldsymbol{a}\}}$ is shown as a bold link.

## 2.2 Relational Expressions and Background Filtering

Note that usually $M \gg N$, so that there are far more records than dimensions (in our example, $M = 74 > 3 = N$). In principle, filters $J$ defining which records to include in a view can be specified arbitrarily, for example through any SQL or MDX `where` clause, or through OLAP operations like `top` $n$, including the $n$ records with the highest value of some feature. In practice, filters are specified as relational expressions in terms of the dimensional values, as expressed in MDX `where` clauses. In our example, we might say `where RPM Mfr = "Ludlum" and ( Month <= "Feb" and Month >= "Jan")`, using chronological order on the Month variable to determine a filter $J$ specifying just those 20 out of the total possible 74 records. For notational purposes, we will therefore sometimes use these relational expressions to indicate the corresponding filters.

Note that each relational filter expression references a certain set of variables, in this case RPM Mfr and Month, denoted as $R \subseteq \mathbb{N}_N$. Compared to our projector $I$, $R$ naturally divides into two groups of variables:

**Foreground:** Those variables in $R^f := R \cap I$ which appear in both the filter expression and are included in the current projection.

**Background:** Those variables in $R^b := R \setminus I$ which appear only in the filter expression, but are not part of the current projection.

The portions of filter expressions involving foreground variables restrict the rows and columns displayed in the OLAP tool. Filtering expressions can have many sources, such as `Show Only` or `Hide`. It is common in full (hierarchical) OLAP to select a collection of siblings within a particular sub-branch of a hierarchical dimension. For example for a spatial dimension, the user within the ProClarity tool might select `All -> USA -> California`, or its children `California ->` Cities, all siblings. But those portions of filter expressions involving background variables do not change which rows or columns are displayed, but only serve to reduce the values shown in cells. In ProClarity, these are shown in the Background pane.

## 2.3 Example

Table 2 shows the results of four chaining operations from our original example in Table 1, including a projection $I = \{1, 2, 3\} \mapsto I' = \{1, 2\}$, a filter using relational expressions, and a filter using a non-relational expression. The bottom right shows a hybrid result of applying both the projector $I' = \{1, 2\}$ and the relational filter expression `where RPM Mfr = "Ludlum" and ( Month <= "Feb" and Month >= "Jan")` . Compare this to the top left, where there is only a quantitative restriction for the same dimensionality because of the use of a background filter. Here $I = \{$ RPM Mfr, Location $\}, R = \{$ RPM Mfr, Month $\}, R^f = \{$ RPM Mfr $\}, R^b = \{$ Month $\}, M' = 20$.

**Table 2.** Results from chaining operations $\mathcal{D}_{\mathbb{N}_N,\mathbb{N}_M} \mapsto \mathcal{D}_{I',J'}$ from the data cube in Table 1. (Top Left) Projection: $I' = \{1,2\}, M' = M = 74$. (Top Right) Filter: $J' =$ `where RPM Mfr = "Ludlum" and ( Month <= "Feb" and Month >= "Jan"),` $M' = 20$. (Bottom Left) Filter: $J'$ determined from top 5 most frequent entries, $M' = 45$. (Bottom Right) $I' = \{1,2\}$ and $J'$ determinued by the relational expression `where RPM Mfr = "Ludlum" and ( Month <= "Feb" and Month >= "Jan")`, $M' = 20$.

| RPM Mfr | Location | $c[I'](\boldsymbol{x})$ | $f[I'](\boldsymbol{x})$ |
|---|---|---|---|
| Ludlum | New York | 11 | 0.150 |
| | Seattle | 24 | 0.325 |
| | Miami | 15 | 0.203 |
| SAIC | New York | 1 | 0.014 |
| | Seattle | 10 | 0.136 |
| | Miami | 13 | 0.176 |

| RPM Mfr | Location | Month | $c^{J'}(\boldsymbol{x})$ | $f^{J'}(\boldsymbol{x})$ |
|---|---|---|---|---|
| Ludlum | New York | Jan | 1 | 0.050 |
| | Seattle | Jan | 9 | 0.450 |
| | Miami | Jan | 2 | 0.100 |
| | | Feb | 8 | 0.400 |

| RPM Mfr | Location | Month | $c^{J'}(\boldsymbol{x})$ | $f^{J'}(\boldsymbol{x})$ |
|---|---|---|---|---|
| Ludlum | Seattle | Apr | 15 | 0.333 |
| | | Jan | 9 | 0.200 |
| | Miami | Feb | 8 | 0.178 |
| | New York | Apr | 7 | 0.156 |
| SAIC | New York | Jan | 6 | 0.133 |

| RPM Mfr | Location | $c^{J'}[I'](\boldsymbol{x})$ | $f^{J'}[I'](\boldsymbol{x})$ |
|---|---|---|---|
| Ludlum | New York | 1 | 0.050 |
| | Seattle | 9 | 0.450 |
| | Miami | 10 | 0.500 |

## 3 Conditional Views

In this section consider the filter $J$ to be fixed, and supress the superscript on $f$. We have seen that the frequencies $f : \mathbf{X} \to [0,1]$ represent joint probabilities $f(\boldsymbol{x}) = f(x_{k^1}, x_{k^2}, \ldots, x_{k^N})$, so that from (2) and (5), $f[I](\boldsymbol{x} \downarrow I)$ expresses the $I$-way marginal over a joint probability distribution $f$. Now consider two projectors $I_1, I_2 \subseteq \mathbb{N}_N$, so that we can define a conditional frequency $f[I_1|I_2] : \mathbf{X} \downarrow I_1 \cup I_2 \to [0,1]$ where $f[I_1|I_2] := \frac{f[I_1 \cup I_2]}{f[I_2]}$. For individual vectors, we have

$$f[I_1|I_2](\boldsymbol{x}) = f[I_1|I_2](\boldsymbol{x} \downarrow I_1 \cup I_2) := \frac{f[I_1 \cup I_2](\boldsymbol{x} \downarrow I_1 \cup I_2)}{f[I_2](\boldsymbol{x} \downarrow I_2)}.$$

$f[I_1|I_2](\boldsymbol{x})$ is the probability of the vector $\boldsymbol{x} \downarrow I_1 \cup I_2$ restricted to the $I_1 \cup I_2$ dimensions given that we know we can only choose vectors whose restriction to $I_2$ is $\boldsymbol{x} \downarrow I_2$. We note that $f[I_1|\emptyset](\boldsymbol{x}) = f[I_1](\boldsymbol{x}), f[\emptyset|I_2] \equiv 1$, and since $f[I_1|I_2] = f[I_1 \setminus I_2|I_2]$, in general we can assume that $I_1$ and $I_2$ are disjoint.

We can now extend our concept of a view to a **conditional view** $\mathcal{D}_{I_1|I_2,J}$ as a view on $\mathcal{D}_{I_1 \cup I_2,J}$, which is further equipped with the conditional frequency $f^J[I_1|I_2]$. Conditional views $\mathcal{D}_{I_1|I_2,J}$ live in a different combinatorial structure than the view lattice $\mathcal{B}$. To describe $I_1|I_2$ and $J$ in a conditional view, we need three sets $I_1, I_2 \in \mathbb{N}_N$ and $J \in \mathbb{N}_M$ with $I_1$ and $I_2$ disjoint. So define $\mathcal{A} := 3^{[N]} \times 2^M$ where $3^{[N]}$ is a graded poset [9] with the following structure:

- $N+1$ levels numbered from the bottom $0, 1, \ldots N$.
- The $i^{th}$ level contains all partitions of each of the sets in $\binom{[N]}{i}$, that is the $i$-element subsets of $\mathbb{N}_N$, into two parts where

1. The order of the parts is significant, so that $[\{1,3\},\{4\}]$ and $[\{4\},\{1,3\}]$ of $\{1,3,4\}$ are not equivalent.
2. The empty set is an allowed member of a partition, so $[\{1,3,4\},\emptyset]$ is in the third level of $3^{[N]}$ for $N \geq 4$.

– We write the two sets without set brackets and with a | separating them.
– The partial order is given by an extended subset relation: if $I_1 \subseteq I_1'$ and $I_2 \subseteq I_2'$, then $I_1|I_2 \prec I_1'|I_2'$, e.g. $1\ 2|3 \prec 1\ 2\ 4|3$.

An element in the poset $3^{[N]}$ corresponds to an $I_1|I_2$ by letting $I_1$ (resp. $I_2$) be the elements to the left (resp. right) of the |. We call this poset $3^{[N]}$ because it's size is $3^N$ and it really corresponds to partitioning $\mathbb{N}_N$ into three disjoint sets, the first being $I_1$, the second being $I_2$ and the third being $\mathbb{N}_N \setminus (I_1 \cup I_2)$. The structure $3^{[2]}$ is shown in Fig. 2.

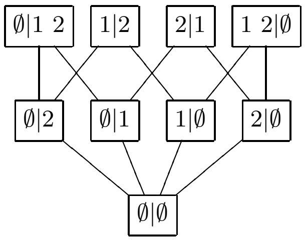


**Fig. 2.** The structure $3^{[2]}$

## 4 Information Measures on Conditional Views

For a view $\mathcal{D}_{I,J} \in \mathcal{B}$ which we identify with its frequency $f^J[I]$, or a conditional view $\mathcal{D}_{I_1|I_2;J} \in \mathcal{A}$ which we identify with its conditional frequency $f^J[I_1|I_2]$, we are interested in measuring how "interesting" or "unusual" it is, as measured by departures from a null model. Such measures can be used for combinatorial search over the view structures $\mathcal{B}, \mathcal{A}$ to identify noteworthy features in the data. The entropy of an unconditional view $\mathcal{D}_{I,J}$

$$H(f^J[I]) := -\sum_{\boldsymbol{x} \in \mathbf{X}\downarrow I} f^J[I](\boldsymbol{x}) \log(f^J[I](\boldsymbol{x})).$$

is a well-established measure of the information content of that view. A view has maximal entropy when every slot has the same expected count. Given a conditional view $\mathcal{D}_{I_1|I_2,J}$, we define the **conditional entropy**, $H(f^J[I_1|I_2])$ to be the expected entropy of the conditional distribution $f^J[I_1|I_2]$, which operationally is related to the unconditional entropy as

$$H(f^J[I_1|I_2]) := H(f^J[I_1 \cup I_2]) - H(f^J[I_2]).$$

Given two views $\mathcal{D}_{I,J}, \mathcal{D}_{I,J'}$ of the same dimensionality $I$, but with different filters $J$ and $J'$, the **relative entropy** (Kullback-Leibler divergence)

$$D(f^J[I] \| f^{J'}[I]) := \sum_{\boldsymbol{x} \in \mathbf{X}\downarrow I} f^J[I](\boldsymbol{x}) \log \left( \frac{f^J[I](\boldsymbol{x})}{f^{J'}[I](\boldsymbol{x})} \right)$$

is a well-known measure of the similarity of $f^J[I]$ to $f^{J'}[I]$. $D$ is zero if and only if $f^J[I] = f^{J'}[I]$, but it is not a metric because it is not symmetric, i.e., $D(f^J[I] \| f^{J'}[I]) \neq D(f^{J'}[I] \| f^J[I])$.

$D$ is a special case of a larger class of $\alpha$-divergence measures between distribution introduced by Csiszár [8]. Given two probability distributions $P$ and $Q$, write the density with respect to the dominating measure $\mu = P = Q$ as $p = dP/d(P+Q)$ and $q = dQ/d(P+Q)$. For any $\alpha \in \mathbb{R}$, the $\alpha$-divergence is

$$D_\alpha(P||Q) = \int \frac{\alpha p(x) + (1-\alpha) q(x) - p(x)^\alpha q(x)^{1-\alpha}}{\alpha(1-\alpha)} \mu(dx).$$

$\alpha$-divergence is convex with respect to both $p$ and $q$, is non-negative, and is zero if and only $p = q$ $\mu$-almost everywhere. For $\alpha \neq 0, 1$, the $\alpha$-divergence is bounded. The limit when $\alpha \to 1$ returns the relative entropy between $P$ and $Q$. There are other special cases that are of interest to us:

$$D_2(P||Q) = \frac{1}{2} \int \frac{(p(x) - q(x))^2}{q(x)} \mu(dx)$$
$$D_{-1}(P||Q) = \frac{1}{2} \int \frac{(q(x) - p(x))^2}{p(x)} \mu(dx)$$
$$D_{1/2}(P||Q) = 2 \int \left( \sqrt{p(x)} - \sqrt{q(x)} \right)^2 \mu(dx).$$

In particular the **Hellinger metric** $\sqrt{D_{1/2}}$ is symmetric in both $p$ and $q$, and satisfies the triangle inequality. We prefer the Hellinger distance over the relative entropy because it is a bonified metric and remains bounded. In our case and notation, we have the **Hellinger distance** as

$$G(f^J[I], f^{J'}[I]) := \sqrt{\sum_{\boldsymbol{x} \in \mathbf{X}\downarrow I} \left( \sqrt{f^J[I](\boldsymbol{x})} - \sqrt{f^{J'}[I](\boldsymbol{x})} \right)^2}.$$

## 5 Hop-Chaining View Discovery

Given our basic formalism on data views, conditional views, and information measures on them, a variety of possible user-guided navigational tasks become possible. For example, above we discussed Cariou *et al.* [5], who develop methods for discovering "drill-down paths" in data cubes. We can describe this as creating a series of views with projectors $I_1 \supseteq I_2 \supseteq I_3$ of increasingly specified dimensional structure.

Our approach is motivated by the idea that most users will be challenged by complex views of high dimensionality, while still needing to explore many possible data interactions. We are thus interested in restricting our users to two-dimensional views only, producing a sequence of projectors $I_1, I_2, I_3$ where $|I_k| = 2$ and $|I_k \cap I_{k+1}| = 1$, thus affecting a permutation of the variables $X^i$.

## 5.1 Preliminaries

We assume an arbitrary permutation of the $i \in \mathbb{N}_N$ so that we can refer to the dimensions $X^1, X^2, \ldots, X^N$ in order. The choice of the initial variables $X^1, X^2$ is a free parameter to the method, acting as a kind of "seed".

One thing that is critical to note is the following. Consider a view $\mathcal{D}_{I,J}$ which is then filtered to include only records for a particular member $x_0^{i_0} \in X^{i_0}$ of a particular dimension $X^{i_0} \in \mathcal{X}$; in other words, let $J'$ be determined by the relational expression `where` $\mathtt{X}^{\mathtt{i}_0} = \mathtt{x}_0^{\mathtt{i}_0}$. Then in the new view $\mathcal{D}'_{I,J'}$, $f^{J'}[I]$ is positive only on the fibers of the tensor $\mathbf{X}$ where $X^{i_0} = x_0^{i_0}$, and zero elsewhere. Thus the variable $X^{i_0}$ is effectively removed from the dimensionality of $\mathcal{D}'$, or rather, it is removed from the *support* of $\mathcal{D}'$.

Notationally, we can say $\mathcal{D}_{I,X^{i_0}=x_0^{i_0}} = \mathcal{D}_{I \setminus \{i_0\},X^{i_0}=x_0^{i_0}}$. Under the normal convention that $0 \cdot \log(0) = 0$, our information measures $H$ and $G$ above are insensitive to the addition of zeros in the distribution. This allows us to compare the view $\mathcal{D}_{I,X^{i_0}=x_0^{i_0}}$ to any other view of dimensionality $I \setminus \{i_0\}$.

This is illustrated in Table 3 through our continuing example, now with the filter `where Location="Seattle"`. Although formally still an RPM Mfr $\times$ Location $\times$ Month cube, in fact this view lives in the RPM Mfr $\times$ Month plane, and so can be compared to the RPM Mfr $\times$ Month marginal.

**Table 3.** Our example data tensor from Table 1 under the filter `where Location="Seattle"`; $M' = 34$

| RPM Mfr | Location | Month | $c(\boldsymbol{x})$ | $f(\boldsymbol{x})$ |
|---|---|---|---|---|
| Ludlum | Seattle | Jan | 9 | 0.265 |
| | | Apr | 15 | 0.441 |
| SAIC | | Feb | 4 | 0.118 |
| | | Mar | 3 | 0.088 |
| | | Apr | 3 | 0.088 |

Finally, some caution is necessary when the relative entropy $D(f^J[I] \| f^{J'}[I])$ or Hellinger distance $G(f^J[I], f^{J'}[I])$ is calculated from data, as their magnitudes between empirical distributions is strongly influenced by small sample sizes. To counter spurious effects, we supplement each calculated entropy with the probability that under the null distribution that the row has the same distribution as the marginal, of observing an empirical entropy larger or equal to actual value. When that probability is large, say greater than 5%, then we consider consider its value spurious and set it to zero before proceeding with the algorithm.

## 5.2 Method

We can now state the hop-chaining methodology.

1. Set the initial filter to $J = \mathbb{N}_M$. Set the initial projector $I = \{1, 2\}$, determining the initial view $f^J[I]$ as just the initial $X^1 \times X^2$ grid.
2. For each row $x_{k^1} \in X^1$, we have the marginal distribution $f^{X^1=x_{k^1}}[I]$ of that individual row, using the superscript to indicate the relational expression filter. We also have the marginal $f^J[I \setminus \{X^1\}]$ over all the rows for the current filter $J$. In light of the discussion just above, we can calculate all the Hellinger distances between each of the rows and this row marginal as
$$G(f^{X^1=x_{k^1}}[I], f^J[I \setminus \{X^1\}]) = G(f^{X^1=x_{k^1}}[I \setminus \{X^1\}], f^J[I \setminus \{X^1\}]),$$
and retain the maximum row value $G^1 := \max_{x_{k^1} \in X^1} G(f^{X^1=x_{k^1}}[I], f^J[I \setminus \{X^1\}])$. We can dually do so for columns against the column marginal:
$$G(f^{X^2=x_{k^2}}[I], f^J[I \setminus \{X^2\}]) = G(f^{X^2=x_{k^2}}[I \setminus \{X^2\}], f^J[I \setminus \{X^2\}]),$$
retaining the maximum value $G^2 := \max_{x_{k^2} \in X^2} G(f^{X^2=x_{k^2}}[I], f^J[I \setminus \{X^2\}])$.
3. The user is prompted to select either a row $x_0^1 \in X^1$ or a column $x_0^2 \in X^2$. Since $G^1$ (resp. $G^2$) represents the row (column) with the largest distance from its marginal, perhaps selecting the global maximum $\max(G^1, G^2)$ is most appropriate; or this can be selected automatically. Letting $x_0'$ be the selected value from the selected variable (row or column) $i' \in I$, then $J'$ is set to $\texttt{where } \mathtt{X}^{\mathtt{i}'} = \mathtt{x}_0'$, and this is placed in the *background* filter.
4. Let $i'' \in I$ be the variable *not* selected by the user, so that $I = \{i', i''\}$.
5. For each dimension $i''' \in \mathbb{N}_N \setminus I$, that is, for each dimension which is neither in the background filter $R^b = \{i'\}$ nor retained in the view through the projector $\{i''\}$, calculate the conditional entropy of the retained view $f^{J'}[\{i''\}]$ against that variable: $H(f^{J'}[\{i''\}|\{i'''\}])$.
6. The user is prompted to select a new variable $i''' \in \mathbb{N}_N \setminus I$ to add to the projector $\{i''\}$. Since $\underset{i''' \in \mathbb{N}_N \setminus I}{\operatorname{argmin}} H(f^{J'}[\{i''\}|\{i'''\}])$ represents the variable with the most constraint against $i''$, that may be the most appropriate selection, or it can be selected automatically.
7. Let $I' = \{i'', i'''\}$. Note that $I'$ is a sibling to $I$ in $\mathcal{B}^N$, thus the name "hop-chaining".
8. Let $I', J'$ be the new $I, J$ and go to step 2.

Keeping in mind the arbitrary permutation of the $X^i$, then the repeated result of applying this method is a sequence of hop-chaining steps in the view lattice, building up an increasing background filter:

1. $I = \{1, 2\}, J = \mathbb{N}_M$
2. $I' = \{2, 3\}, J' = \texttt{where } \mathtt{X}^1 = \mathtt{x}_0^1$
3. $I'' = \{3, 4\}, J'' = \texttt{where } \mathtt{X}^1 = \mathtt{x}_0^1, \mathtt{X}^2 = \mathtt{x}_0^2$
4. $I''' = \{4, 5\}, J''' = \texttt{where } \mathtt{X}^1 = \mathtt{x}_0^1, \mathtt{X}^2 = \mathtt{x}_0^2, \mathtt{X}^3 = \mathtt{x}_0^3$

### 5.3 Extension to Hierarchical Data Cubes

In Sec. 2.1 we introduced data cubes as flat data tensors, while in general in OLAP the dimensions are hierarchically structured. While a full development of a hierarchical approach to hop-chaining awaits future work, we can indicate those directions here. First, we extend our definition of a data cube to be $\mathcal{D} := \langle \mathbf{X}, \mathcal{X}, \mathcal{Q}, c \rangle$, where now $\mathcal{Q} = \{\mathcal{P}^i\}_{i=1}^N$ is a collection of $N$ partially-ordered hierarchical [9] **dimensions** $\mathcal{P}^i = \langle P^i, \leq^i \rangle$ with **members** $p^i \in P^i$. Each partially-ordered set (poset) $\mathcal{P}^i$ is isomorphic to a sub-poset of the Boolean lattice $\mathcal{B}^i = \left\langle 2^{X^i}, \subseteq \right\rangle$ which is the power set of the values of the variable $X^i$ ordered by set inclusion. While in principle each poset $\mathcal{P}^i$ could be as large as $\mathcal{B}^i$, in practice, they are trees, with $X^i \in P^i$ and $\forall x^i \in X^i, \{x^i\} \in P^i$.

We can identify the **cube schema** as $\mathbf{P} := \bigtimes_{i=1}^N \mathcal{P}^i$, so that each $\boldsymbol{p} \in \mathbf{P}$ is a **cell**. We then have the hierarchical count function $\hat{c} : \mathbf{P} \to \mathbb{N}$, where $\hat{c}(\boldsymbol{p}) := \sum_{\boldsymbol{x} \leq \boldsymbol{p}} c(\boldsymbol{x})$, and $\leq := \bigtimes_{i=1}^N \leq^i$, the product order of the hierarchies. There is also the hierarchical frequency function $\hat{f} : \mathbf{P} \to [0,1]$, with $\hat{f}(\boldsymbol{p}) := \frac{\hat{c}(\boldsymbol{p})}{M}$. In a real OLAP view, the entries actually correspond not to slots $\boldsymbol{x} \in \mathbf{X}$, but to cells $\boldsymbol{p} \in \mathbf{P}$; and the rows and columns not to collections of data items $Y^i \subseteq X^i$, but of members $Q^i \subseteq P^i$. If $X^1 =$ "Location", and $p_0^1 =$ "California" $\in P^1$, then classical drilldown might take a row like `California` from a view, restrict $J$ with the relational expression `where Location` $=$ `California`, and then replace $Q^1$ with all the children of $p_0^1$, so that $Q'^1 = \{p^1 \leq^1 p_0^1\}$.

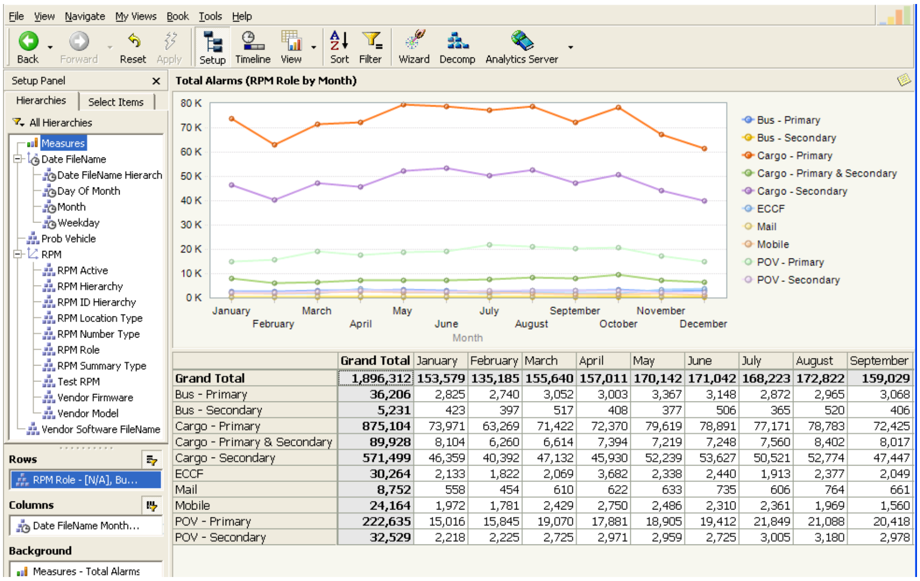
We are experimenting with view discovery and hop-chaining formalisms which operate over these member collections $Q^i$, and in general over their Cartesian products $\bigtimes_{i \in I} Q^i \subseteq \mathbf{P} \downarrow I$. But in the current formulation, it is sufficient to consider each dimension $X^i$ involved in a foreground view to be drilled-down to the immediate children of the top of $\mathcal{P}^i$, that is, the children of `All`.

## 6 Implementation

We have implemented the hop-chaining methodology in a prototype form for experimentation and proof-of-principle. ProClarity 6.2 is used in conjunction with SQL Server Analysis Services (SSAS) 2005 and the R statistical platform v. 2.7[6]. ProClarity provides a flexible and friendly GUI environment with extensive API support which we use to gather current display contents and query context for row, column and background filter selections. R is currently used in either batch or interactive mode for statistical analysis and development. Microsoft Visual Studio .Net 2005 is used to develop plug-ins to ProClarity to pass ProClarity views to R for hop-chain calculations.

A first view of the data set used for demonstrating this method is shown in Fig. 3, a screenshot from the ProClarity tool. The database is a collection of 1.9M records of events of RPM events. The 15 available dimensions are shown on the left of the screen (e.g. "day of the month", "RPM hierarchy"), tracking such

[6] `http://www.r-project.org`



| | Grand Total | January | February | March | April | May | June | July | August | September |
|---|---|---|---|---|---|---|---|---|---|---|
| **Grand Total** | **1,896,312** | **153,579** | **135,185** | **155,640** | **157,011** | **170,142** | **171,042** | **168,223** | **172,822** | **159,029** |
| Bus - Primary | **36,206** | 2,825 | 2,740 | 3,052 | 3,003 | 3,367 | 3,148 | 2,872 | 2,965 | 3,068 |
| Bus - Secondary | **5,231** | 423 | 397 | 517 | 408 | 377 | 506 | 365 | 520 | 406 |
| Cargo - Primary | **875,104** | 73,971 | 63,269 | 71,422 | 72,370 | 79,619 | 78,891 | 77,171 | 78,783 | 72,425 |
| Cargo - Primary & Secondary | **89,928** | 8,104 | 6,260 | 6,614 | 7,394 | 7,219 | 7,248 | 7,560 | 8,402 | 8,017 |
| Cargo - Secondary | **571,499** | 46,359 | 40,392 | 47,132 | 45,930 | 52,239 | 53,627 | 50,521 | 52,774 | 47,447 |
| ECCF | **30,264** | 2,133 | 1,822 | 2,069 | 3,682 | 2,338 | 2,440 | 1,913 | 2,377 | 2,049 |
| Mail | **8,752** | 558 | 454 | 610 | 622 | 633 | 735 | 606 | 764 | 661 |
| Mobile | **24,164** | 1,972 | 1,781 | 2,429 | 2,750 | 2,486 | 2,310 | 2,361 | 1,969 | 1,560 |
| POV - Primary | **222,635** | 15,016 | 15,845 | 19,070 | 17,881 | 18,905 | 19,412 | 21,849 | 21,088 | 20,418 |
| POV - Secondary | **32,529** | 2,218 | 2,225 | 2,725 | 2,971 | 2,959 | 2,725 | 3,005 | 3,180 | 2,978 |

**Fig. 3.** Initial 2-D view of the alarm summary data cube, showing count distribution of RPM Role by months

things as the identities and characteristics of particular RPMs, time information about events, and information about the hardware, firmware, and software used at different RPMs.

## 7 Examples

Space limitations will allow showing only a single step for the hop-chaining procedure against the alarm summary data cube.

Fig. 3 shows the two-dimensional projection of the $X^1$ = "RPM Role" $\times X^2$ = "Month" dimensions within the 15-dimensional overall cube, drilled down to the first level of the hierarchies (see Sec. 5.3). Its plot shows the distributions of

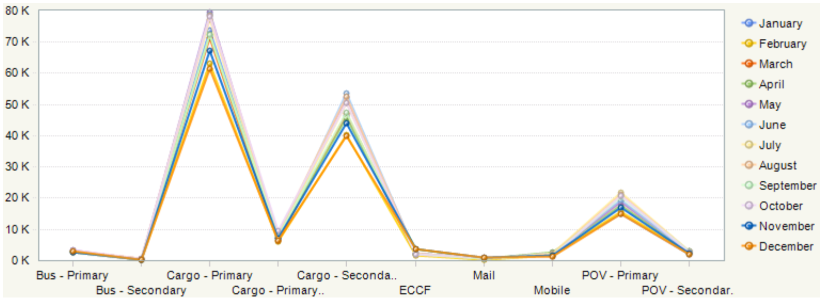


**Fig. 4.** Count distribution of months

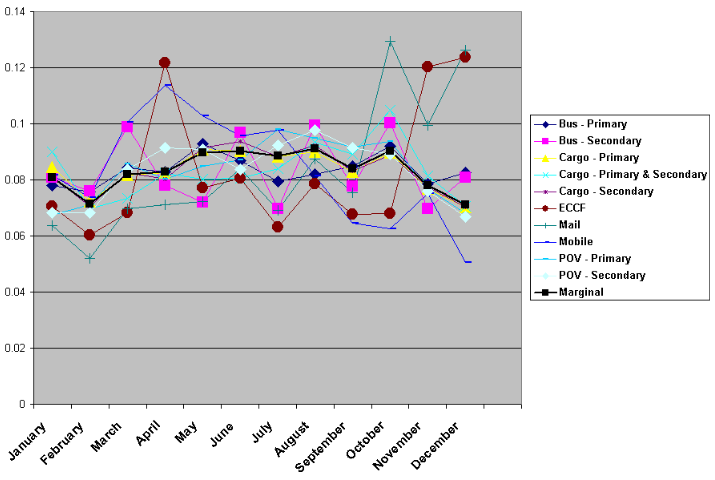


**Fig. 5.** Frequency distributions of RPM roles

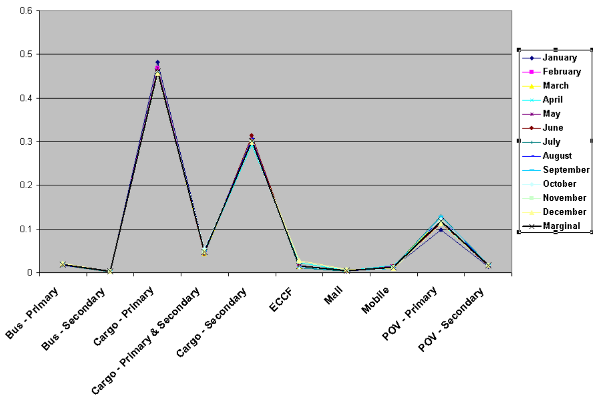


**Fig. 6.** Frequency distributions of months

count $c$ of alarms by RPM role (Busses Primary, Cargo Secondary, etc.) $X^1$, while Fig. 4 shows the distribution by Month $X^2$.

The distributions for roles seem to vary at most by overall magnitude, rather than shape, while the distributions for months appear almost identical. However, Fig. 5 and Fig. 6 show the same distributions, but now in terms of their frequencies $f$ relative to their corresponding marginals, allowing us to compare the shapes of the distributions normalized by their absolute sizes. While the months still seem identical, the RPM roles are clearly different, although it is difficult to see which is most unusual with respect to the marginal (bold line).

The left side of Fig. 7 shows the Hellinger distances $G(f^{X^i=x_{k^i}}[I], f^J[I \setminus \{X^i\}])$ for $i \in \{1, 2\}$ for each row or column against its marginal. The RPM roles "ECCF" and "Mail" are clearly the most significant, which can be verified by examining the anomolously shaped plots in Fig. 5. The most significant

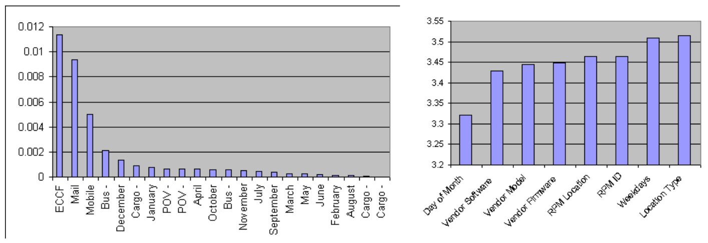
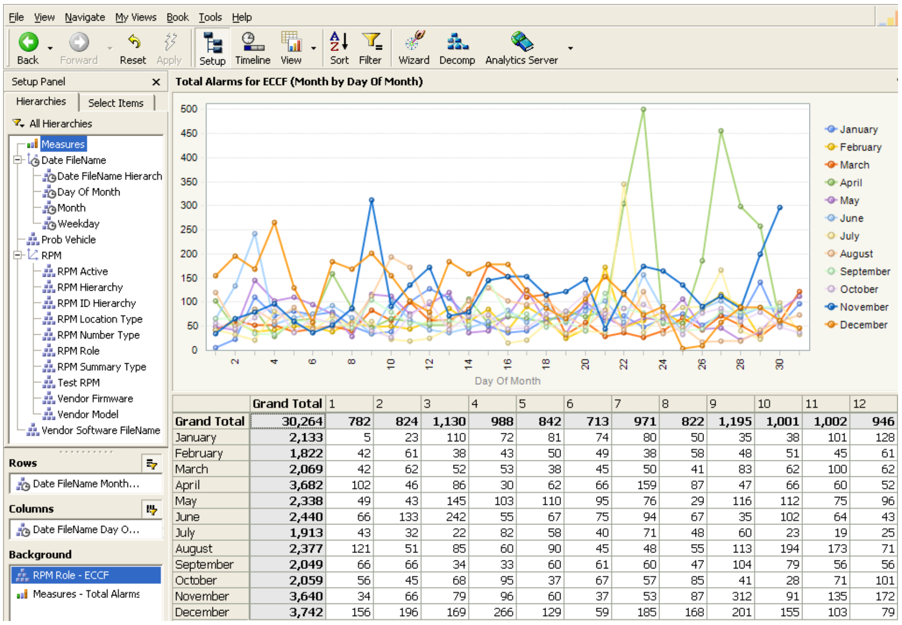


**Fig. 7.** (Left) Hellinger distances of rows and columns against their marginals. (Right) Relative entropy of months against each other significant dimension, given that RPM Role = ECCF.



| | Grand Total | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Grand Total** | **30,264** | **782** | **824** | **1,130** | **988** | **842** | **713** | **971** | **822** | **1,195** | **1,001** | **1,002** | **946** |
| January | **2,133** | 5 | 23 | 110 | 72 | 81 | 74 | 80 | 50 | 35 | 38 | 101 | 128 |
| February | **1,822** | 42 | 61 | 38 | 43 | 50 | 49 | 38 | 58 | 48 | 51 | 45 | 61 |
| March | **2,069** | 42 | 62 | 52 | 53 | 38 | 45 | 50 | 41 | 83 | 62 | 100 | 62 |
| April | **3,682** | 102 | 46 | 86 | 30 | 62 | 66 | 159 | 87 | 47 | 66 | 60 | 52 |
| May | **2,338** | 49 | 43 | 145 | 103 | 110 | 95 | 76 | 29 | 116 | 112 | 75 | 96 |
| June | **2,440** | 66 | 133 | 242 | 55 | 67 | 75 | 94 | 67 | 35 | 102 | 64 | 43 |
| July | **1,913** | 43 | 32 | 22 | 82 | 58 | 40 | 71 | 48 | 60 | 23 | 19 | 25 |
| August | **2,377** | 121 | 51 | 85 | 60 | 90 | 45 | 48 | 55 | 113 | 194 | 173 | 71 |
| September | **2,049** | 66 | 66 | 34 | 33 | 60 | 61 | 60 | 47 | 104 | 79 | 56 | 56 |
| October | **2,059** | 56 | 45 | 68 | 95 | 37 | 67 | 57 | 85 | 41 | 28 | 71 | 101 |
| November | **3,640** | 34 | 66 | 79 | 96 | 60 | 37 | 53 | 87 | 312 | 91 | 135 | 172 |
| December | **3,742** | 156 | 196 | 169 | 266 | 129 | 59 | 185 | 168 | 201 | 155 | 103 | 79 |

**Fig. 8.** Subsequent view on the $X^2$ = Months $\times X^3$ = Day of Month projector. Note the new background filter `where RPM Role = ECCF`.

month is December, although this is hardly evident in Fig. 6. We select the maximal row-wise Hellinger value $G^1 = .011$ for ECCF, so that $i' = 1, x_0^1 =$ ECCF. $X^{i'} = X^1 =$ "RPM Role" is added to the backgound filter, $X^{i''} = X^2$ = Months is retained in the view, and we calculate $H(f^{J'}[\{2\}|\{i'''\}])$ for all $i''' \in \{3, 4, \ldots, 15\}$, which are shown on the right of Fig. 7 for all significant dimensions. On that basis $X^3$ is selected as Day of Month with minimal $H = 3.22$.

The subsequent view for $X^2$ = Months $\times X^3$ = Day of Month is then shown in Fig. 8. Note the strikingly divergent plot for April: it in fact does have the highest Hellinger distance at .07, an aspect which is completely invisible from the overall initial view, e.g. in Fig. 5.

## 8 Discussion, Analysis, and Future Work

In this paper, we have provided the fundemantals necessary to express view discovery in OLAP databases as a combinatorial search and optimization operation in general, aside from the specific hop-chaining method. What remains to be addressed is a precise formal expression of this optimization problem. This is dependent on the mathematical properties of our information measures $H, D$, and $G$ over the lattices $\mathcal{B}, \mathcal{A}$. It is well known, for example, that $H$ is a monotonic function in $\mathcal{B}$ [9], in that $\forall I_1 \subseteq I_2, H(f^J[I_1]) \geq H(f^J[I_2])$. There should be ample literature (e.g. [27]) to fully explicate the behavior of these functions on these structures, and guide development of future search algorithms.

Also as mentioned above, we are restricting our attention to OLAP cubes with a single "count" measure. Frequency distributions are available from other quantitative measures, and exploring the behavior of these algorithms in those contexts is of interest.

Information measures are used because of their mathematical properties in identifying unusual distributions. It remains to be demonstrated that these measures have high utility for real analysts of these databases, and which mix of statistical measures, whether including our precise hop-chaining algorithm or not, is ideal for their needs.

The value of our implementation within commercial front-end database tools provides the opportunity for this validation. Generally, software implementations provide a tremendous value in performing this research, not only for this practical validation by sponsors and users, but also for assisting with the methodological development itself. As our software platform matures, we look forward to incorporating other algorithms for view discovery [5,16,20,23,24,25], for purposes of comparison and completeness.

## Acknowledgements

This work is supported in part by the U.S. Department of Homeland Security under Grant Award 2007-ST-104-000006. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security. We are grateful for the support of Judith Cohn at the Los Alamos National Laboratory; David Gillen, Eric Stephan, Bryan Olsen, and Elena Peterson at the Pacific Northwest National Laboratory; Matt Damante and Robert Fernandez at the Lawrence Livermore National Laboratory; and our sponsors and colleagues at DHS/S&T, DHS/ICE, and DHS/CBP.

## References

1. Agrawal, R., Gupta, A., Sarawagi, S.: Modeling Multidimensional Databases. In: Proc. 13th Int. Conf. on Data Engineering (1997)
2. Asimov, D.: The Grand Tour: A Tool for Viewing Multidimensional Data. SIAM J. Statistical Computing 6, 1 (1985)
3. Barak, B., Chaudhuri, K., Dwork, C., Kale, S., McSherry, F., Talwar, K.: Privacy, Accuracy, and Consistency Too: A Holistic Solution to Contingency Table Release. In: Proc. 2007 Conf. Principles of Database Systems (PODS 2007) (2007)
4. Borgelt, C., Kruse, R.: Graphical Models. Wiley, New York (2002)
5. Cariou, V., Cubillé, J., Derquenne, C., Goutier, S., Guisnel, F., Klajnmic, H.: Built-In Indicators to Discover Interesting Drill Paths in a Cube. In: Song, I.-Y., Eder, J., Nguyen, T.M. (eds.) DaWaK 2008. LNCS, vol. 5182, pp. 33–44. Springer, Heidelberg (2008)
6. Chaudhuri, S., Umeshwar, D.: An Overview of Data Warehousing and OLAP Technology. ACM SIGMOD Record 26(1), 65–74 (1997)
7. Codd, E.F., Codd, S.B., Salley, C.T.: Providing OLAP (On-Line Analitycal Processing) to User-Analysts: An IT Mandate" (1993), `www.cs.bgu.ac.il/~dbm031/dw042/Papers/olap_to_useranalysts_wp.pdf`
8. Csiszár, I.: Information-type measures of divergence of probability distributions and indirect observations. Studia Sci. Math. Hung 2, 299–318 (1967)
9. Davey, B.A., Priestly, H.A.: Introduction to Lattices and Order, 2nd edn. Cambridge University Press, Cambridge (1990)
10. Gyssens, M., Lakshmanan, L.V.S: A Foundation for Multi-Dimensional Databases. In: Proc. 23rd VLDB Conf., pp. 106–115 (1997)
11. Joslyn, C.A., Gillen, D., Fernandes, R., Damante, M., Burke, J., Critchlow, T.: Hybrid Relational and Link Analytical Knowledge Discovery for Law Enforcement. In: 2008 IEEE Int. Conf. on Technologies for Homeland Security (HST 2008), pp. 161–166. IEEE, Piscataway (2008)
12. Joslyn, C., Mniszeiski, S.: DEEP: Data Exploration through Extension and Projection. Los Alamos Technical Report LAUR 02-1330 (2002)
13. Klir, G., Elias, D.: Architecture of Systems Problem Solving, 2nd edn. Plenum, New York (2003)
14. Kolda, T.G., Bader, B.W.: Tensor Decompositions and Applications. SIAM Review (in press) (2008)
15. Krippendorff, K.: Information Theory: Structural Models for Qualitative Data. Sage Publications, Newbury Park (1986)
16. Kumar, N., Gangopadhyay, A., Bapna, S., Karabatis, G., Chen, Z.: Measuring Interestingness of Discovered Skewed Patters in Data Cubes. Decision Support Systems 46, 429–439 (2008)
17. Lauritzen, S.L.: Graphical Models, Oxford UP (1996)
18. Malvestuto, F.M.: Testing Implication of Hierarchical Log-Linear Models for Probability Distributions. Statistics and Computing 6, 169–176 (1996)
19. Montanari, A., Lizzani, L.: A Projection Pursuit Approach to Variable Selection. Computational Statistics and Data Analysis 35, 463–473 (2001)
20. Palpanas, T., Koudas, N.: Using Database Aggregates for Approximating Querying and Deviation Detection. IEEE Trans. Knowledge and Data Engineering 17(11), 1–11 (2005)
21. Pedersen, T.B., Jensen, C.S.: Multidimensional Database Technology. IEEE Computer 34(12), 40–46 (2001)

22. Sarawagi, S.: Explaining Differences in Multidimensional Aggregates. In: Proc. 25th Int. Conf. Very Large Databases, VLDB 1999 (1999)
23. Sarawagi, S.: User-Adaptive Exploration of Multidimensional Data. In: Proc. 26th Very Large Database Conf., pp. 307–316 (2000)
24. Sarawagi, S.: iDiff: Informative Summarization of Differences in Multidimensional Aggregates. Data Mining and Knowledge Discovery 5, 255–276 (2001)
25. Sarawagi, S., Agrawal, R., Megiddo, N.: Discovery-driven Exploration of OLAP Data Cubes. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 168–182. Springer, Heidelberg (1998)
26. Soshani, A.: OLAP and Statistical Databases: Simlarities and Differences. In: Proc. PODS 1997, pp. 185–196 (1997)
27. Studeny, M.: Probabilistic Conditional Independence Structures. Springer, London (2005)
28. Thomas, H., Datta, A.: A Conceptual Model and Algebra for On-Line Analytical Processing in Decision Support Databases. Information Systems Research 12(1), 83–102 (2001)

# Designing a Geo-scientific Request Language - A Database Approach

Peter Baumann

Jacobs University Bremen, Campus Ring 12,
28759 Bremen, Germany
p.baumann@jacobs-university.de

**Abstract.** A large part of sensor, image, and statistics data in the Earth Sciences naturally come as data points aligned to regular grids of some dimension, including 1-D time series, 2-D imagery, 3-D image time series and volumetric data, and 4-D spatio-temporal data. Frequently repositories have to host objects of multi-Terabyte size, in the future multi-Petabytes. Serving this information category, large multi-dimensional arrays, is a task of increasing importance among the Earth Sciences. Still, however, ad-hoc implementations with focused functionality prevail which lack the flexibility of SQL-enabled databases.

The Web Coverage Processing Service (WCPS) geo raster model and language allows navigation, extraction, and ad-hoc analysis on multi-dimensional geoscientific raster data which can be geo-referenced or not. The request language has been designed to offer some key properties considered advantageous by the database community: it is formalized, declarative, data independent, optimizable, and safe in evaluation. WCPS has been adopted as an international standard by the Open GeoSpatial Consortium (OGC) in December 2008. The reference implementation is almost finished. Actually, the embedding into the modular framework of the OGC geo service standards has posed particular constraints which the design had to respect. We discuss conceptual model, query language, and the context using real-life use case scenarios.

**Keywords:** geo services, raster databases, scientific databases, query languages, standardization.

## 1 Introduction

A large part of sensor, image, and statistics data in the earth sciences naturally come as data points aligned to regular grids of some dimension. For example, in-situ sensors deliver 1-D time series data. Remote sensing instruments on board of satellites produce 2-D imagery, often with hundreds of bands. For many analysis and documentation tasks it is convenient to arrange such images into 3-D x/y/t image time series. Geophysical data often are 3-D by nature resulting in x/y/z datacubes; atmospheric, oceanographic, and cosmologic simulations result, e.g., in 4-D spatiotemporal x/y/z/t data sets.

While 1-D time series usually impress more by their large number of objects arising than by their individual sizes, data volumes immediately grow very large; multi-Gigabyte objects are considered small, multi-Terabyte sizes are common today, and multi-Petabyte objects are on the horizon.

Although servicing this information category, large multi-dimensional arrays, represent a typical database task current database technology does not do much to support it. Consequently, today ad-hoc implementations prevail which often are tuned towards specific, simple tasks, but do not offer the flexibility known from SQL-enabled databases.

This information category, large multi-dimensional arrays, today usually is served by ad-hoc implementations which often are tuned towards specific, simple tasks, but do not offer the flexibility known from SQL-enabled databases. One reason is that the standard relational model does not give adequate support. Normally images as such are more or less treated as ancillary data and abandoned as soon as relevant information has been pre-extracted, such as in multimedia databases. If stored in the database then the only concept offered is BLOBs [12] which come with no advanced retrieval functionality.

However, Earth Science researchers increasingly demand for out-of-the-box software allowing to serve a variety of observation or simulation data to their own communities, but also to other communities or even the general public. Such services need to have both expert level (like versatile search, retrieval, and analysis tools) and easy-to-use generic interfaces (like GoogleMaps style visually oriented interactive data navigation). As the history of SQL shows, query languages can well serve a plurality of interface styles through one common underlying operational model. Hence, it seems adequate to pursue such a concept for versatile raster data support in the Earth Sciences as well.

Background of the work reported here is the work on open, interoperable geo service interfaces in the context of the Open GeoSpatial Consortium (OGC)[1] where the author co-chairs the raster-relevant working groups on WCS (Web Coverage Service), WCPS (Web Coverage Processing Service), and Coverages (a Discussion Working Group which does not develop specifications, but serves as a topical discussion and exchange forum).

"Coverage" is the standard term for *spatio-temporally extended phenomena* a in its broad sense, but in current practice narrowed down to raster data. On such coverages, the OGC Web Coverage Service (WCS) standard [24] defines a service interface to retrieve coverage data and metadata. For raster data retrieval, the GetCoverage request type supports spatio-temporal and band subetting, scaling, reprojection, and data format encoding for shipping results. WCS tentatively offers a limited functionality to ease uptake and use[2]. More than once requests were brought to the WCS Working Group to incorporate additional processing functionality. The group ultimately decided that it is not beneficial to add isolated specific functionality when requested. One reason is that such functionality frequently lacks a uniformly accepted definition, but exists in a variety of more or less diverging incarnations without one agreed canonical representative. Another reason is that such a list of functions is open ended, and responding to focused requests in the end would lead to a large, unstructured set of unrelated functions.

[1] www.opengeospatial.org, last seen on April 11, 2009.

[2] Actually it turns out that handling of different coordinate systems and their transformation, for example, imposes quite some challenges.

Therefore, the approach adopted in the end is to provide a high-level, interoperable processing language based on the existing WCS data model which allows clients to phrase their individual functionality as requests which are understood by any conformant server. Building on the rich experience in designing and standardizing retrieval languages made by the database community it seemed natural to apply their techniques for achieving properties like flexibility, declarativeness, optimizability, and safe evaluation. Additionally, concepts from language design – such as XQuery and object-oriented languages – and image processing have been incorporated. On the other hand, stream processing was expressly excluded, as the prevailing use cases for the moment being demand but evaluation of complete preexisting data sets, assumed static at the time of query execution as in traditional databases. A streaming extension might be added in future once sufficient need is conceived.

The result of this activity is the OGC Web Coverage Processing (WCPS) Language Interface Standard [4] together with its WCS protocol embedding, the WCS Processing Extension Interface Standard [5], developed by the author and his group and adopted as an official standard by OGC in December 2008.

In the remainder of this paper, we first present the pre-existing conceptual model of coverages within OGC in Section 2. On this basis, we present of design rationales and relevant related work in Section 3. In Section 4, the query model is laid out, followed by thoughts on service embedding in Section 5. In Section 6, sample application scenarios are sketched. Section 7 addresses implementation issues, and Section 8 presents summary and outlook.

## 2 Conceptual Model

The conceptual model of WCPS relies on the WCS definition of coverages; this in turn is based on ISO 19123 [10] and OGC Abstract Specification Topic 6 [17], which, in fact, are mutually agreed and identical. Coverages are categorized there into discrete and continuous types. Discrete coverages are subdivided further into *Discrete Point Coverage*, *Discrete Grid Point Coverage*, *Discrete Curve Coverage*, *Discrete Surface Coverage*, and *Discrete Solid Coverage*. The continuous coverage type has subclasses *Thiessen Polygon*, *Hexagonal Grid*, *Segmented Curve*, *Continuous Quadrilateral Grid*, and *TIN* (Triangular Irregular Network). Out of this comprehensive list, current standards address only coverages of type Discrete Grid Point, which in essence are raster data. The reason is that currently only these are understood well enough to allow for comprehensive and stable standardization. Further coverage types are already under consideration for service standardization, though.

A WCS coverage object [24] consists of a raster array and a few metadata elements. The raster part essentially represents a function mapping d-dimensional points from some *domain* (which in our case is spatio-temporal) to some *range* (often known as "pixel" or "voxel" data type). Domains define the region in which the *cells* sit (which are regularly spaced in case of our raster-type coverages), each containing a range value. A domain of some dimensionality *d* has an (unordered!) set of *d axes*, identified by name. Four axis names are reserved and have the well-known special semantics: *x*, *y*, *z*, and *t*. In future, so-called *abstract axes* will be allowed in addition which can represent a domain semantics beyond the predefined spatio-temporal axes. For example, in atmospheric and ocean modelling pressure often is used as an axis, and sometimes a second time axis.

Domains are expressed in some *Coordinate Reference System* (CRS) which is defined by a coordinate system and a datum. OGC uses the nomenclature of the European Petroleum Standards Group (EPSG) which defines several thousands CRSs, most of them in active use by the communities. Along each axis the extent of a coverage is defined by some closed interval, for each CRS. Overall, therefore, coverage domains form some axis-parallel hypercube in the respective coordinate system space. Often CRSs address only a subset of coordinate axes; for example, the widely used WGS-84 can express x/y and x/y/z coordinate spaces, but not time. For time values, ISO 8601:2000 is available as CRS. In this case, a suitable combination of CRSs must be used for addressing a particular coverage cell. Only so-called Image CRSs allow to address cells in integer coordinates, independent from their physical location reference. Each coverage knows about the CRSs in which its domain can be addressed; among them must be at least one Image CRS.

This brief illustration of the geographic domain concept illustrates already that an appropriate conceptual modelling is all but trivial, and by no means consists just of a pure, integer-addressable cell array.

Neither are range values trivial. WCPS allows a subset of the structures currently available with WCS. The type of a WCPS range value can be an atomic base type or a (currently non-nested) record of named *range fields*; the latter resemble the concept of "bands" or "channels" common in hyperspectral image processing. Base types available are the numeric types available in standard programming languages (char, short, int, long in signed and unsigned variants, plus float and double) and, additionally complex numbers of single and double precision.

Values which bear a null semantics are listed in a – possibly empty – set of null values associated with each coverage. Further, for each range field a – possibly empty – set of those interpolation methods is listed which can be applied to this coverage range field during any operation that requires value interpolation, such as scaling and reprojection. Among the possible interpolation methods are *nearest neighbor*, *linear*, *quadratic*, and *cubic*. An associated property named *interpolation null resistance* specifies how null values are treated during interpolation.

In brief, a WCS coverage consists of the following elements [24]:

- A string-valued identifier which is unique on a given server;
- A domain description, consisting of axes, admissible CRSs, and extent per axis;
- A range type description, consisting of fields, their types, null values, interpolation methods and null resistance;
- A multi-dimensional array of values, one per domain coordinate cell;
- Some further metadata which are of no concern for our discussion.

## 3 Design Rationales

WCPS is the outcome of several years of discussion with relevant players, potential users, and OGC standards implementers. Prior to developing the specification (and sometimes in parallel, as new developments appeared) some design considerations were made based on a review of the state of the art. In this section we summarize design rationales and preexisting approaches which had impact on WCPS design.

## 3.1 Requirements

The currently envisaged use of WCPS can be summarized as navigation, extraction, and server-side analysis over large repositories of large, multi-dimensional coverages. *Navigation* of coverage data requires capabilities at least like WMS (meaning subsetting, scaling, overlaying, and styling), but on objects of different dimensionalities and often without an intrinsic visual nature (such as elevation or classification data). Versatile portrayal and rendering capabilities, therefore, play an important role. *Extraction and download* involve tasks like retrieving satellite image bands, performing band combinations, or deriving vegetation index maps and classification; hence, they likewise require subsetting, summarization, and processing capabilities. *Analysis* mainly includes multi-dimensional spatiotemporal statistics. In summary, a range of imaging, signal processing, and statistical operations should be expressible; some of these aspects have been studied in [8].

Additionally, the language should not be too distant in its conceptual model from existing geo data processing tools (such as the ones listed in the next section) so that it is economically feasible for vendors to implement the standard as an additional layer on top of their existing products. On a side note, still such implementations obviously can differ widely in terms of performance, scalability, domain support, and other factors.

Further, it should be possible for some deployed service to accept new, unanticipated request types without extra programming, in particular not on server side. The rationale behind is, while that both lay users and experts frequently come up with new desires, it is not feasible for a service provider to continuously invest into programming of new service functionality. Ideally the service interface paradigm offers open-ended expressiveness available without client-side or server-side programming. This calls for a language approach where users (or client developers) can flexibly combine existing building blocks into new functionality.

From databases we learn that it is advantageous to craft such a language in a way that it is safe and declarative. Safe evaluation ensures that no Denial of Service (DoS) attack is possible on the level of a single request. In languages like SQL this is achieved by avoiding explicit loop and recursion constructs; what this means for the handling of arrays where everybody tends to think about iteration loops in the first place will have to be discussed also in the context of declarativeness. As is well known safe evaluation requires a trade-off against expressiveness; we consciously maintain that the processing language be safe in evaluation, retaining operations like convolutions, but losing, e.g., matrix inversion, genetic algorithms, and neural networks.

Declarativeness not just eases request formulation, but also forms a prerequisite for optimization. As our experience with array databases tells that there is a wide range of effective optimization methods on coverage manipulation [20], optimizability of expressions is an important requirement.

Notably we do not demand minimality of the language; while the number of operations certainly should be kept as small as possible, we decided to rank usability higher.

Coverage support no longer is constrained to 2-D imagery and 3-D image timeseries. Since some time now coverages are perceived as 1-D to 4-D entities in discussion, and "abstract", i.e., non-spatiotemporal axes have been brought up, such as atmospheric pressure. Hence, coverage expressions should allow to freely walk through the dimensions, in any combination of spatial, temporal, and abstract axes. For example, 2-D coverages with $x$ and $z$ axes can well occur as slicing results from 3-D or 4-D coverages. On a side note, such considerations in the course of WCPS design actually to some extent have had impact on the WCS coverage model and, eventually, also have contributed to a generalization of OGC's model of coordinate systems.

Further, the language should be Semantic Web ready in that coverage access and manipulation is described in a completely machine-readable form, independent from human intervention when it comes to service discovery and orchestration.

Finally, given that an international standard is aiming at a large and diverse community and stands to assure semantic interoperability between heterogeneous clients and servers, a formal specification of syntax and semantics seems indispensable. Still, the resulting specification document needs to be understandable, in particular to programmers not necessarily familiar with mathematical semantics definition. While the many attempts of combining both properties in a model have shown that this seems close to impossible, a suitable compromise should be aimed at.

On a side note, ease of comprehension also rules out a pure XML encoding; languages like XQuery and XPath show how compact language style can be combined with XML.

## 3.2 Candidates

As databases traditionally do not support large rasters, earth scientists tend to build their own, ad-hoc systems. These usually come with bespoke, procedural interfaces and without clear conceptual models in a database sense.

For the conceptual design of a coverage processing language suitable for use in a Web environment we have evaluated existing OGC standards, image processing, and image and array databases.

**OGC WPS.** *Web Processing Service* (WPS) specification is an OGC standard which specifies a geo service interface for any sort of GIS functionality across a network [20]. A WPS may offer calculations as simple as subtracting one set of spatially referenced numbers from another (e.g., determining the difference in influenza cases between two different seasons), or as complicated as a global climate change model. This model makes WPS especially suitable for "webifying" legacy applications. Essentially, this boils down to a remote method invocation in the spirit of RPC, CORBA, and Java RMI, but additionally with explicit geospatial semantics in the XML schema. As such, it brings along all the concerns of, e.g., XML-RPC[3], such as the severe security compromise.

[3] www.xmlrpc.com, last seen on April 11, 2009.

Another grave argument is the low semantic level of WPS. To understand this better let us inspect an example. A server-side routine provides a function Buffer[4] which accepts an input polygon and returns a buffered feature. In the corresponding service description which is based on the standardized WPS XML Schema (see Figure 1), function name as well as input and output parameters are described. This represents the function signature, i.e., operation syntax; looking for the semantics specification we find XML elements *Title* and *Abstract* containing human-readable text. Hence, there is no way for any automated agent to use it without human interference at some point.

This has a number of serious drawbacks:

- WPS consists only of a low-level syntactic framework for procedural invocation without any coverage specific operations; In other words, the processing functionality itself is not specified, hence any high-level services implemented on top of WCS per se are not standardized and interoperable\footnote{the WPS specification already mentions that it requires specific profiles to achieve fully-automated interoperability.};
- XML offers only syntactic service interoperability, as opposed to the semantic interoperability of WCPS; adding any new functionality to a WPS installation requires new programming on both client and server side; such a service cannot be detected by an automatic agent, as the semantics is not machine understandable; for the same reason, automatic service chaining and orchestration cannot be achieved - for example, it is unclear to an automatic agent how to connect output parameters of one processing step with the input parameters of the next step due to the missing semantics information;
- with a similar argument, server-interal optimization such as dynamic load balancing is difficult at least.

Hence, for our purpose the low-level WPS model needs to be complemented by some high-level coverage processing model.

**Image Processing.** We immediately rule out computer vision and image understanding, as these disciplines work on a different semantic level than WCPS is aiming at. Further, answers generated in these domains normally are of probabilistic nature, whereas for WCPS the goal is to allow precise responses whenever possible. Many image processing languages have been proposed, such as MatLab[5], Erdas Imagine[6], and ENVI[7]. These to some extent imperative languages offer a wide range of proven functionality. However, they are not safe in evaluation and often also not declarative, thereby lacking optimization potential in a database sense. The underlying systems usually do not have a satisfactory storage management suitable for Petabyte-size objects.

[4] In GIS terminology, a *buffer* is an area of specified distance around some object, for example, a 100m strip left and right to some highway.

[5] www.mathworks.com, last seen on April 1122, 2009.

[6] gi.leica-geosystems.com/LGISub1x33x0.aspx, last seen on April 11, 2009.

[7] www.rsinc.com/envi, last seen on April 11, 2009.

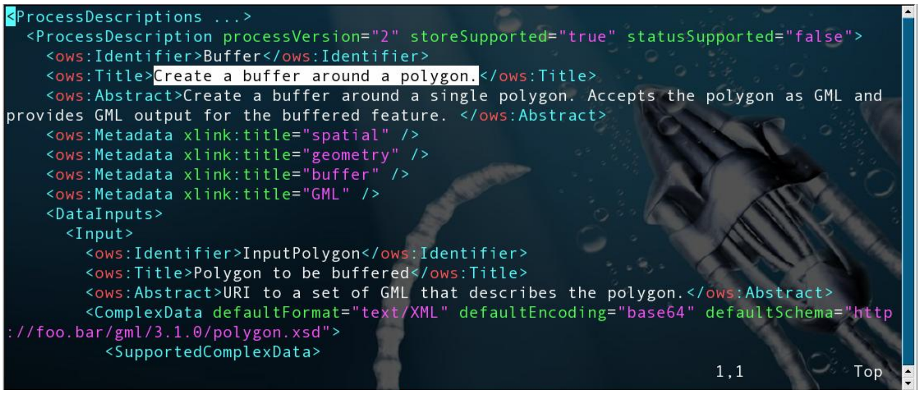

```
<ProcessDescriptions ...>
  <ProcessDescription processVersion="2" storeSupported="true" statusSupported="false">
    <ows:Identifier>Buffer</ows:Identifier>
    <ows:Title>Create a buffer around a polygon.</ows:Title>
    <ows:Abstract>Create a buffer around a single polygon. Accepts the polygon as GML and
provides GML output for the buffered feature. </ows:Abstract>
    <ows:Metadata xlink:title="spatial" />
    <ows:Metadata xlink:title="geometry" />
    <ows:Metadata xlink:title="buffer" />
    <ows:Metadata xlink:title="GML" />
    <DataInputs>
      <Input>
        <ows:Identifier>InputPolygon</ows:Identifier>
        <ows:Title>Polygon to be buffered</ows:Title>
        <ows:Abstract>URI to a set of GML that describes the polygon.</ows:Abstract>
        <ComplexData defaultFormat="text/XML" defaultEncoding="base64" defaultSchema="http
://foo.bar/gml/3.1.0/polygon.xsd">
          <SupportedComplexData>
                                                                              1,1          Top
```

**Fig. 1.** Excerpt from a WPS process specification

**Image Databases.** The requirement for further coverage processing capabilities distinguishes array databases from multimedia databases. In multimedia databases images are interpreted using some hand-picked built-in algorithm to obtain feature vectors; subsequently, search is performed on the feature vectors, not the images. The WCPS language, conversely, does not attempt to interpret the coverage data, but always operates on the cell values themselves. Hence, search results are not subject to a probability and are not depending on some random, hidden interpretation algorithm.

Another distinguishing criterion, albeit on architectural level, is the potentially large amount of data which implementations of the standard need to process efficiently. Image processing systems traditionally are constrained to image sizes less than main memory available; to some extent, "out of memory" algorithms have been designed which essentially perform partial access and swapping of parts.

**Array databases.** A systematic approach to processing large multi-dimensional raster data volumes is addressed by the research field of array databases. Several conceptual frameworks, including partial or complete implementations, have been proposed, such as (chronologically) rasdaman [1,2,17], AQL [12], AML [14,16], and RAM [23]. Studying their query functionality offers good hints for the envisaged sensor, image, and statistics processing. However, it turns out that the notion of arrays is too low-level for the purpose on hand, due to several shortcomings.

First, in all array formalisms the axes are ordered and accessed by their position. This is not adequate for spatio-temporal models where axis ordering is no explicit feature, quite the opposite: row-major or column-major ordering of images is a physical property which ideally should not affect query formulation; actually, data format encodings internally tend to not uniformly agree on their internal linearization methods, which makes it necessary that the axis semantics is known at encoding time.

Additionally, the spatio-temporal axes – let us them call *x*, *y*, *z*, and *t* – have a well-defined axis semantics which has several implications on processing. For example, images normally are scaled in *x* and *y* simultaneously, and coordinate system transformations apply on *x* and *y* axes. Addressing along the *t* axis, on the other hand, is conceptually independent from space coordinates and expressed in completely different coordinate systems, such as ISO 8601 [8].

**Industrial implementations.** Marketed Web-capable geo raster servers include ESRI's ArcSDE[8] and Oracle's GeoRaster cartridge[9]. Both are rather restricted in their data models and access capabilities. Oracle, for example, supports only 2-D raster images, only selected cell types, and no integration of retrieval capabilities into SQL *select* or *where* statements. ArcSDE does not offer any retrieval language at all. The only commercial array DBMS with full query support is rasdaman[10].

## 4 Request Language

Based on the coverage model sketched above, a functional language on coverages has been developed. The specification style adopted for WCPS is semi-formal, in an attempt to combine diverging requirements. From a technical perspective, only a formal specification allows to prove relevant properties like consistency and sufficient completeness; further, it usually aids implementors by being crisp and concise; the prose form often chosen for geo service specifications has well-known issues and, hence, was considered inadequate. On the other hand, consumers of standards often have neither background nor inclination to immerse into algebra, logics, and the like; a pure mathematical specification like, for example, the excellently crafted ISO standard LOTOS [11], would not be accepted by the target communities. In the end, the style chosen follows a clear, formally tractable structure, but gets informal when it comes to concepts whose formalization would take excessive formalism and is well-known to the community anyway, such as image reprojection.

Similar to the algebraic specification of Abstract Data Types, coverage functions are described via their effect observable through applying the other functions. To this end, first a set of 17 probing functions is established which determine each relevant property of a coverage object. For each operation, then, its semantics is defined through the changes that can be observed by applying the probing functions.

For example, coverage probing function *imageCrsDomain*(*C*,*a*) delivers the extent (i.e, lower and upper bound) of coverage *C* on axis *a*. Another function, *value*(*C*,*p*), delivers the range value of the cell with address *p* in coverage *C*.

In the sequel we will given an overview of the language and describe selected operations in detail to give a flavour of the specification style.

A WCPS request can address any number of coverages known to the server addressed. The result is a sequence of either coverages or scalar values. The general structure of a WCPS query is given as follows:

```
for var_1 in ( cov_1,1, …, cov_1,n1 ),
    …,
    var_n in ( cov_n,1, …, cov_n,nm )
[ where booleanExpr( var_1, …, var_n ) ]
return
    processExpr( var_1, …, var_n )
```

[8] www.esri.com, last seen on April 11, 2009.
[9] www.oracle.com, last seen on April 11, 2009.
[10] www.rasdaman.com, last seen on April 11, 2009.

Literals $\texttt{cov}_{i,j}$ denote the names of coverages available on the server addressed. Variable names $\texttt{var}_i$ are bound to each coverage in turn, following a nested loop pattern with $\texttt{var}_1$ governing the outermost loop and $\texttt{var}_n$ the innermost one. Optional predicate `booleanExpr`, which may contain occurrences of all variables, acts as a filter: only if the predicate is fulfilled for a particular coverage combination then the subsequent `return` clause will be executed to contribute an element to the overall result list.

For a coverage-valued result, the outermost function of a return clause must be an `encode()` function. It takes a coverage and encodes it in the format passed as second parameter, returning an ancoded data stream. For example, the following query delivers coverages A, B, and C encoded in TIFF:

```
for $c in ( A, B, C )
return
    encode( $c, "TIFF" )
```

Actually, there is one exception to having the encoding function outermost: Function `store()` can be applied to the encoded result, with the effect that the result is a URI reference to the result data stored in files on the server for subsequent download. This is actually the only side effect of the language.

The first argument to the encoding function is an arbitrarily nested expression which generates a coverage. This can mainly be a coverage variable, a subsetting expression, an induce expression, a CRS transformation expression, a coverage constructor expression, or a condense expression.

Subsetting allows to extract sub-coverages of the same dimensionality by using a `trim` operator or of lower dimensionality using the `slice` operator. Induced operations apply some operation simultaneously to all cells of a given coverage, whereby these operations are those given by the coverage range type and include the well-known arithmetic, trigonometric, exponential functions, plus the cast operator known from programming languages. For example, the following query extracts a time slice from some data cube C having at least a time axis and delivers the log of each cell value:

```
for $c in ( C )
return
    encode( log( slice( $c , t( 0 ) ) ), "NetCDF" )
```

The trim example shall serve to illustrate the specification style. The corresponding excerpt is listed in the Appendix.

A coverage constructor allows to generate a new coverage "from scratch" by defining its axes, extent per axis, and an expression for calculating a value for each cell. We will see an example in conjunction with the condense operator.

The condenser is a generalization of the relational aggregation. It accepts a scalar expression and iterates over it while aggregating all values according to some commutative and associative value type operation. The syntax is

```
condense op
over     var₁ axisType₁ ( extent₁ ),
         …,
         varₙ axisTypeₙ ( extentₙ ),
using    scalarExpr
```

For each combination of ($\texttt{var}_1$ ,… , $\texttt{var}_n$) `scalarExpr` is evaluated, and the result aggregated via operation `op` until the final result is returned. Convenient shorthand versions are defined for the frequent simple cases that no cell address airthmetic needs to be performed. For example, `count(B)` counts the *true* elements in some boolean coverage expression `B`, determines `max(C)` determines the maximum value of coverage expression `C`, and `some(B)` checkes whether there is at least one element in boolean expression `B`.

This allows us to formulate a practically motivated example of both coverage constructor and condenser. Let `C` be a coverage expression over cell type `unsigned char`. Then, the following expression derives a histogram of `C`:

```
coverage histogram
over     bucket x ( 0 : 255 )
values   count( C = bucket )
```

Coverage histogram of axis type `x`, is defined to range from 0 to 255, with variable `bucket` assuming each value in turn. Induced operation "=" compares `C` against the current value of `bucket`, and `count()` finds out how many equalities occur.

This recursively defined language allows arbitrary nesting of coverage expressions. Additionally, type compatibility and extension are provided for convenience and efficiency by allowing to avoid frequent explicit cast operations.

As it has been shown in [2], the coverage constructor and the condense operator can describe all other coverage expressions (except CRS transforms). Still, further operations are provided for convenience and because these turn out to be particularly suitable for optimization [20].

## 5 Service Embedding

The syntax described above is deliberately kept independent from any protocol, hence it is also called "Abstract Syntax". This allows to embed the language into any target environment. In the framework of OGC standardization, two such environments, WCS and WPS, are foreseen currently.

The first concrete embedding is with the WCS suite. WCS currently defines three request types, *GetCapabilities*, *DescribeCoverage*, and *GetCoverage*. Following the common OGC service pattern, a *GetCapabilities* request allows a client to retrieve information from a server on its capabilities and its data sets offered. The *DescribeCoverage* request type is specific to WCS; for a given coverage identifier it allows to retrieve detail information (i.e., metadata) about the coverage on hand. A *GetCoverage* request, finally, is used to extract a subset of some coverage served. The *OGC WCS Processing Extension Interface Standard* [5] adds a fourth request type, *ProcessCoverages*, which enables a client to submit a WCPS request and receive the processing results. A WCPS query can be sent by using either HTTP GET with key/value pair (KVP) notation or HTTP POST in conjunction with an XML encoding. The WCPS response is a (possibly empty) list of either coverages or scalars. Coverages are returned using the same response format as a

*GetCoverage* request, thereby minimizing complexity. Scalars are returned as newline separated strings; for the future an XML encoding is foreseen.

As WCPS touches both WCS and WPS terrain the question has been debated repeatedly whether this functionality is better included in WCS or WPS. Indeed there may be applications which put more emphasis on the processing aspect, possibly on further data types, and hence are interested in WCPS, but not WCS. To support such use case scenarios a WPS embedding is under work. This *WPS Profile*, as it is called following WPS terminology, defines a WCPS service as a processing function with a query input parameter and a set-valued output parameter.

## 6 Reference Implementation

The WCPS Reference Implementation is developed at Jacobs University, based on the rasdaman array DBMS [17]. Figure 2 shows the overall system architecture. The WCPS server frontend receives queries in string or XML encoding as defined in [5], analyzes them for syntactic and semantic correctness based on the metadata stored in relational tables, and generates a corresponding query in the rasdaman query language, rasql. The rasdaman server is the actual workhorse which prepares the query response based on the data stored in the relational database as well. Some (currently rather simplistic) clients are available for displaying 1-D time series results, 2-D imagery, and 3-D voxel cubes.

Coverages are mapped to the rasdaman model as follows. The conceptual model of rasdaman [72] consists of unordered collections (following ODMG terminology [71])

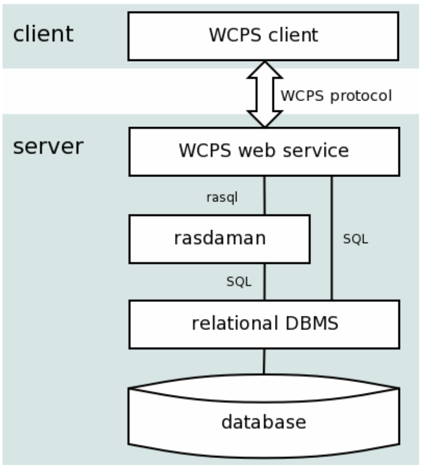


**Fig. 2.** WCPS Reference Implementation system architecture

of (OID,A) pairs where OID is a system-generated object identifier and A is an array. By convention, for each coverage a separate (single-tuple) collection is created which bears the coverage's name.

For the additional metadata making up a coverage a relational schema is provided. The WCPS frontend uses these metadata for a semantic check and for generating the corresponding rasql query. Due to the conceptual similarity of WCPS and rasql this mapping is relatively straightforward. For example, the histogram request shown previously, when embedded into a complete request with an assumed data format of CSV (comma-separated values), translates to this rasql query:

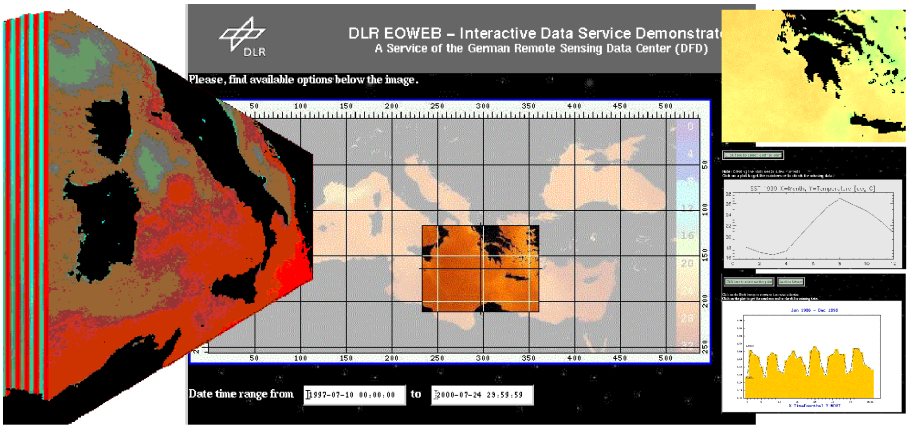
```
select csv( marray bucket in [ 0 : 255 ]
            values count_cells( (C) = bucket )
          )
from   C
```

Aside from the different syntax, the WCPS frontend in this example resolves the named axes with coordinate reference system based addressing to positional axes with integer addressing.

The rasdaman server executes such queries on raster objects stored in partitions called *tiles* [71]. In the optimization phase [20] the server attempts to apply techniques like algebraic rewriting, pre-aggregate exploitation, parallelization, and just-in-time compilation for CPU or GPU. Finally, the query is evaluated employing *tile streaming* [25], i.e., tile-by-tile evaluation, which in many cases allows materializing only one tile at a time in the server's main memory. By construction of the mapping, each result of the set-oriented query contains exactly one result array. This result is encoded in the requested format and passed back to the WCPS frontend which ships it to the client.

## 7 Application Scenarios

A WCPS demonstration site is in progress under www.earthlook.org. It allows queries on 1-D to 4-D earth science data sets from the application scenarions sensor data analysis, remote sensing, oceanography, geophysics, and atmospheric simulation. Here we demonstrate WCPS in two hand-picked use cases: a 3-D image time series application and a 4-D life science scenario.


DLR
DLR EOWEB - Interactive Data Service Demonstrat
A Service of the German Remote Sensing Data Center (DFD)
Please, find available options below the image.
50 100 150 200 250 300 350 400 450 500
50 100 150 200 250
Jan 1996 - Dec 1998

Date time range from 1997-07-10 00:00:00 to 2000-07-24 23:59:59

**Fig. 3.** DFD-DLR demonstrator for 1-D to 3-D extraction from image time series data cubes (source: DFD-DLR)

### 7.1 Satellite Image Time Series Services

In a precursor project conducted together with the German Aerospace Agency (DFD/DLR) a Web service on x/y/t image time series data cubes has been developed. Among the data sets provided is an AVHRR sea/land surface temperature map. Images have been composed into a seamless map of Europe and the Mediterranean, and then extended over time. The resulting image cube totals around 10,000 images.

Figure 3 (center) shows the Web interface which allows spatial selection via an overview map and temporal selection by time range. The left image represents a 3-D cutout obtained through spatio-temporal subsetting. To the right there is a 2-D time slice (top), a 1-D temperature curve (middle), and a 1-D vegetation index curve (bottom). Such derived data form a typical WCPS application scenario; the Normalized Difference Vegetation Index is computed from multi-spectral satellite images as (*nearInfrared-red*)/(*nearInfrared+red*).

As WCPS was not available at that time this retrieval interface is based on the rasdaman raster query language, whereby queries are generated from a Web interface using IDL on th Net. A temperature curve in the time interval between `T0` and `T1` could be obtained through a WCPS expression like this one:

```
for $t in ( TemperatureCube )
return
    encode(
        $t[ t( T0, T1 ) ],
        CSV
    )
```

### 7.2 Gene Research Data Services

The last example demonstrates application of spatiotemporal raster services beyond the earth sciences. In the ESTEDI project[11] one of the areas investigated was gene expression simulation. There the Mooshka database [19] was established containing gene activation maps of the *drosophila melanogaster* (fruit fly). Each such map, representing activity of a particular gene in a particular fruit fly embryo at a particular point of development, consists of a 3-D voxel map of expression intensities. In geo service terminology, such voxel images are rectified but not georeferenced. Hence, they can be modeled as coverages having no geodetic coordinate system, but only their image coordinate reference system.

Figure 3 shows the classical way of obtaining analysis data: 2-D slices obtained from staining embryos are combined into an RGB image. Choice of the three genes to be combined is arbitrary and driven by the comparisons to be made; the number of three is determined by the channels available in RGB. Spatial aggregations allow to obtain intensity diagrams.

With a multi-dimensional raster service these steps from just one special case in the plurality of possible analysis variants. The following WCPS query assumes a 4-D image where the time axis represents the exemplar's lifetime, always starting with 0. The query delivers average activity of the Hunchback gene over lifetime encoded in CSV (comma-separated values). It does so by generating a new coverage with only a

[11] www.estedi.org, last seen on April 11, 2009.

time axis where each value in this timeline is determined by averaging over the hunchback component of the corresponding time slice.

```
for $e in ( ThisEmbryo )
return
    encode(
        coverage averageActivity
        over     pt t( imageCrsDomain($e,t) )
        values   avg( $e.hunchback[ t( pt ) ] ),
        CSV
    )
```

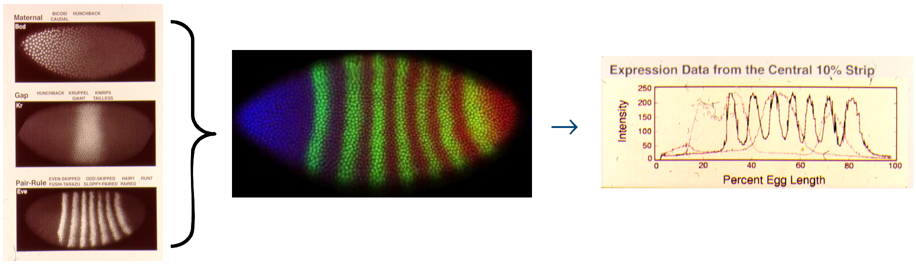


**Fig. 4.** Conventional processing steps in gene expression analysis

## 8 Conclusion

The Web Coverage Processing Service (WCPS) geo raster model and language allows navigation, extraction, and ad-hoc analysis on multi-dimensional geoscientific raster data. The request language has been designed to offer some key properties considered advantageous by the database community:

- It has a (semi-) formal semantics which combines concise syntax and semantics specification with legibility.
- The language is declarative in that there is no explicit array iteration, thereby allowing to process arrays in any cell iteration sequence, in particular based on partitioned ("tiled") storage schemes.
- This in turn opens up a wide space for optimizations of proven effectiveness.
- Coverages are treated in a data independent way: not only are requests independent from data encoding, but also dimensions are addressed by name and not by index, thereby avoiding an artificial dimension sorting.
- WCPS queries are safe in evaluation – every requests terminates after a finite number of steps (proof omitted here, but straightforward).

The interested reader may have noted that the syntax is close to that of XQuery. The idea behind this choice was that, while XQuery does not play any role in coverage/raster services, it might be an interesting research avenue to merge both languages with the goal of integrating semi-structured and raster data retrieval. The issues behind appeared serious enough to not impede the timely roll-out of WCPS by such currently not relevant questions.

WCPS has been adopted as an international standard by the Open GeoSpatial Consortium (OGC) in December 2008, and the reference implementation is almost finished. Next steps include finalization of the WCPS Reference Implementation and release of its source code. Further, an application spectrum as broad as possible will be addressed to evaluate WCPS usability and to gain feedback from the user communities. One line of current research addresses extension of the concepts from equidistant grids to general meshes.

In the end, hope is that WCPS will not only contribute a new quality of service on spatio-temporal raster data, but to foster cross-discpline data exchange and use.

## References

1. Baumann, P.: On the Management of Multidimensional Discrete Data. VLDB Journal 4(3) (1994); Special Issue on Spatial Database Systems, VLDB, 401–444
2. Baumann, P.: A Database Array Algebra for Spatio-Temporal Data and Beyond. In: Tsur, S. (ed.) NGITS 1999. LNCS, vol. 1649, pp. 76–93. Springer, Heidelberg (1999)
3. Baumann, P.: Large-Scale Raster Services: A Case for Databases (invited keynote). In: Roddick, J.F., Benjamins, V.R., Si-said Cherfi, S., Chiang, R., Claramunt, C., Elmasri, R.A., Grandi, F., Han, H., Hepp, M., Lytras, M.D., Mišić, V.B., Poels, G., Song, I.-Y., Trujillo, J., Vangenot, C. (eds.) ER Workshops 2006. LNCS, vol. 4231, pp. 75–84. Springer, Heidelberg (2006)
4. Baumann, P. (ed.): Web Coverage Processing Service (WCPS) Language Interface Standard. OGC 08-068r2 (2009)
5. Baumann, P. (ed.): WCS Processing Extension. OGC 08-059r3 (2009)
6. Catell, R., Cattell, R.G.G.: The Object Data Standard, 3rd edn. Morgan Kaufmann, San Francisco (2000)
7. Furtado, P.: Storage Management of Multidimensional Arrays in Database Management Systems. PhD thesis, Technische Universität München (1999)
8. Garcia, A., Baumann, P.: Modeling Fundamental Geo-Raster Operations with Array Algebra. In: IEEE Intl. Workshop in Spatial and Spatio-Temporal Data Mining, Omaha, USA, October 28-31 (2007)
9. International Organization for Standardization (ISO): Data elements and interchange formats — Information interchange — Representation of dates and times. ISO IS 8601:2000 (2000)
10. International Organization for Standardization (ISO): Geographic Information — Coverage Geometry and Functions. ISO IS 19123:2005 (2005)
11. International Organization for Standardization (ISO): LOTOS, A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. ISO/IEC IS 8807 (1987)
12. Libkin, L., Machlin, R., Wong, L.: A Query Language for Multidimensional Arrays: Design, Implementation, and Optimization Techniques. In: Proc. 15th ACM SIGMOD, Montreal, Canada, pp. 228–239 (1996)
13. Lorie, R.: Issues in Databases for Design Transactions. In: Encarnaçao, J., Krause, F.L. (eds.) File Structures and Databases for CAD. North-Holland, IFIP (1982)
14. Maier, D., Vance, B.: A Call to Order. In: Proc. 12th ACM PODS, Washington, DC, USA, pp. 1–16 (1993)
15. Marathe, A., Salem, K.: A Language for Manipulating Arrays. In: Proc. 23rd VLDB, Athens, Greece, pp. 46–55 (1997)

16. Marathe, A., Salem, K.: Query Processing Techniques for Arrays. In: Proc. SIGMOD, New York, USA, pp. 323–334 (1999)
17. The rasql Query Guide, version 7. rasdaman GmbH (2008)
18. OGC (ed.): Abstract Specification Topic 6: Schema for Coverage Geometry and Functions. OGC 07-011 (2007)
19. Pisarev, A., Poustelnikova, E., Samsonova, M., Baumann, P.: Mooshka: A System for the Management of Multidimensional Gene Expression Data in Situ. Information Systems 28, 269–285 (2003)
20. Ritsch, R.: Optimization and Evaluation of Array Queries in Database Management Systems. PhD thesis, TU Muenchen (2002)
21. Ritter, G., Wilson, J., Davidson, J.: Image Algebra: An Overview. Computer Vision, Graphics, and Image Processing 49(1), 297–336 (1994)
22. Schut, P. (ed.): Web Processing Service (WPS). OGC 05-007r7 (2007)
23. van Ballegooij, A.: Ram: A Multidimensional Array Dbms. In: Lindner, W., Mesiti, M., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) EDBT 2004. LNCS, vol. 3268, pp. 154–165. Springer, Heidelberg (2004)
24. Whiteside, A., Evans, J. (eds.): Web Coverage Service (WCS) Implementation Specification. OGC 07-067r5 (2008)
25. Widmann, N., Baumann, P.: Efficient Execution of Operations in a DBMS for Multidimensional Arrays. In: Proc. 10th Statistical and Scientific Database Management (SSDBM), Capri, Italy, pp. 155–165 (1998)

## Appendix: Sample WCPS Operator Specification

The following is a verbatim sample taken from the WCPS standard [4] specifying spatial subsetting. "Changed?" indicates, for the reader's convenience, which coverage constituent is actually changed by the operation on hand.

Let

$C_1$ be a **coverageExpr**,
$n$ be an **integer** with $0 \leq n$,
$a_1, \ldots, a_n$ be pairwise distinct **dimensionName**s with $a_i \in$ dimensionNameSet($C_1$) for $1 \leq i \leq n$,
$crs_1, \ldots, crs_n$ be **crsName**s with $crs_i \in$ crsList($C_1$) for $1 \leq i \leq n$,
$(lo_1:hi_1), \ldots, (lo_n:hi_n)$ be **dimensionIntervalExpr**s with $lo_i \leq hi_i$ for $1 \leq i \leq n$.

Then,

for any **coverageExpr** $C_2$
where $C_2$ is one of

$C_{\text{bracket}} = C_1$ **[** $p_1$**,** … **,** $p_n$ **]**
$C_{\text{func}} =$ **trim (** $C_1$, **{** $p_1$**,** … **,** $p_n$ **} )**

with

$p_i$ is one of
$p_{\text{img},i} = a_i$ **(** $lo_i$ **:** $hi_i$ **)**
$p_{\text{crs},i} = a_i : crs_i$ **(** $lo_i$ **:** $hi_i$ **)**

$C_2$ is defined as follows:

| **Coverage constituent** | **Changed?** |
|---|---|
| identifier($C_2$) = "" (empty string) | **X** |
| for all $p \in$ imageCrsDomain($C_2$):<br>value($C_2$, $p$ ) = value($C_1$,$p$) | |
| imageCrs($C_2$) = imageCrs($C_1$) | |
| dimensionList($C_2$) = dimensionList($C_1$) | |
| for all $a \in$ dimensionList($C_2$):<br>if $a = a_i$ for some $i$<br>thenimageCrsDomain($C_2$ , $a$ ) = ($lo_{i,img}$ : $hi_{i,img}$ )<br>else imageCrsDomain($C_2$ , $a$ ) = imageCrsDomain($C_1$,$a$)<br>where ($lo_{i,img}$ : $hi_{i,img}$ ) = ($lo_i$:$hi_i$) if no CRS is indicated, and the transform from $crs_i$ into the image CRS if $crs_i$ is indicated. | **X** |
| for all $a \in$ dimensionList($C_2$):<br>crsSet($C_2$, $a$) = crsSet($C_1$, $a$)<br>dimensionType($C_2$, $a$) = dimensionType($C_1$, $a$) | |
| for all $a \in$ dimensionList($C_2$), $c \in$ crsSet($C_2$):<br>if $a = a_i$ for some $i$<br>then domain($C_2$ , $a$, $c$ ) = ($lo_{i,c}$ : $hi_{i,c}$ )<br>else domain($C_2$ , $a$, $c$ ) = domain($C_1$ , $a$, $c$ )<br>where ($lo_{i,c}$:$hi_{i,c}$) represent the dimension boundaries ($lo_i$:$hi_i$) transformed from the $C_2$ image CRS into CRS $c$. | **X** |
| for all $r \in$ rangeFieldNames($C_2$):<br>rangeFieldType( $C_2$, $r$ ) = rangeFieldType($C_1$, $r$) | |
| nullSet($C_2$) = nullSet($C_1$) | |
| for all $r \in$ rangeFieldNames($C_2$ ):<br>interpolationDefault($C_2$, $r$) = interpolationDefault($C_1$, $r$) | |
| for all $r \in$ rangeFieldNames($C_2$ ):<br>interpolationSet($C_2$, $r$) = interpolationSet($C_1$, $r$) | |

# SEEDEEP: A System for Exploring and Querying Scientific Deep Web Data Sources

Fan Wang and Gagan Agrawal

Department of Computer Science and Engineering
Ohio State University, Columbus OH 43210
{wangfa,agrawal}@cse.ohio-state.edu

**Abstract.** A recent and emerging trend in scientific data dissemination involves online databases that are hidden behind query forms, thus forming what is referred to as the *deep web*. In this paper, we propose SEEDEEP, a System for Exploring and quErying scientific DEEP web data sources. SEEDEEP is able to automatically mine deep web data source schemas, integrate heterogeneous data sources, answer cross-source keyword queries, and incorporates features like caching and fault-tolerance. Currently, SEEDEEP integrates 16 deep web data sources in the biological domain. We demonstrate how an integrated model for correlated deep web data sources is constructed, how a complex cross-source keyword query is answered efficiently and correctly, and how important performance issues are addressed.

## 1 Introduction

A recent and emerging trend in scientific data dissemination involves online databases that are hidden behind query forms, thus forming what is referred to as the *deep web* [1]. Recently, hundreds of large, complex, and in many cases, related and/or overlapping, deep web data sources have become available, especially in the scientific domain. The number of such data sources is still increasing rapidly every year [2]. Thus, there is an increasing need for an automatic system that is able to manage and integrate heterogenous scientific deep web data sources. We need to be able to facilitate exploration and queries on these deep web data sources. The following motivating example further illustrates the need of such a system.

**Motivating Example:** A biologist interested in SNP issues a query Q1={ERCC6, SNPID,"ORTH BLAST",HGNCID}, with the following intent: given a gene name `ERCC6`, we want to find the `SNPIDs`, the `BLAST` results and the corresponding `HGNCID` of gene `ERCC6`. To find the `SNPIDs` of `ERCC6`, we need to use gene name `ERCC6` as input to query on `dbSNP` data source to find `SNPIDs`. To find the `BLAST` information, we need to take the following three steps: 1) use `ERCC6` as input to query on `Gene` data source to obtain the proteins of `ERCC6` in human species and other orthologous species; 2) use `Protein` data source to find the sequences of the proteins obtained from step 1; 3) use the protein sequences from step 2 and the SNP information (amino acid position) obtained from `dbSNP` as input to do an alignment using `Entrez BLAST` data source. To find the `HGNCID` of `ERCC6`, we use `ERCC6` as input to query on `HGNC` data source. The plan of answering $Q1$ is shown in Figure 1.

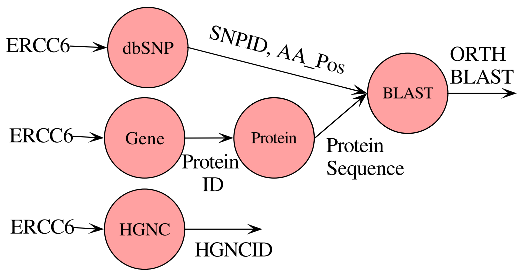


**Fig. 1.** The Plan of Answering Query Q1

From this example, we can see that to answer such a query, the biologist must be able to complete the following steps:

**Step 1:** To learn the complete input and output schemas of biological deep web data sources so that she can *manually identify* appropriate data sources that could be used to answer the query;

**Step 2:** To understand the dependence relation among data sources so that she can *manually figure out* the order by which data sources should be accessed;

**Step 3:** To *manually submit* online queries to numerous query forms and *keep track of* the obtained results;

**Step 4:** To *manually merge and organize* the results from numerous data sources together in a systematic way.

The above procedure for answering *one query* is tedious and error-prone. For a biologist who may issue a number of such queries every day, a system that can automate this process will be highly desirable.

In this paper, we propose SEEDEEP, an automatic system for exploring and querying scientific deep web data sources. Distinct from existing deep web mining systems in the e-commerce domain [3,4,5,6], which mainly focus on schema matching, SEEDEEP has the following features. First, SEEDEEP can mine complete schemas of deep web data sources. Second, SEEDEEP is able to discover the dependence relation among data sources. Third, SEEDEEP can generate multiple query plans for a cross-source keyword query. Finally, SEEDEEP incorporates systems-oriented features like support for data caching and fault-tolerance.

The rest of the paper is organized as follows. In Section 2, we show the system infrastructure of SEEDEEP. We introduce the design of each component in SEEDEEP in Section 3. We describe the implementation of SEEDEEP and show a case study on SEEDEEP in Section 4 and conclude in Section 5.

## 2 System Infrastructure

In this section, we describe the system infrastructure of SEEDEEP which is shown in Figure 2.

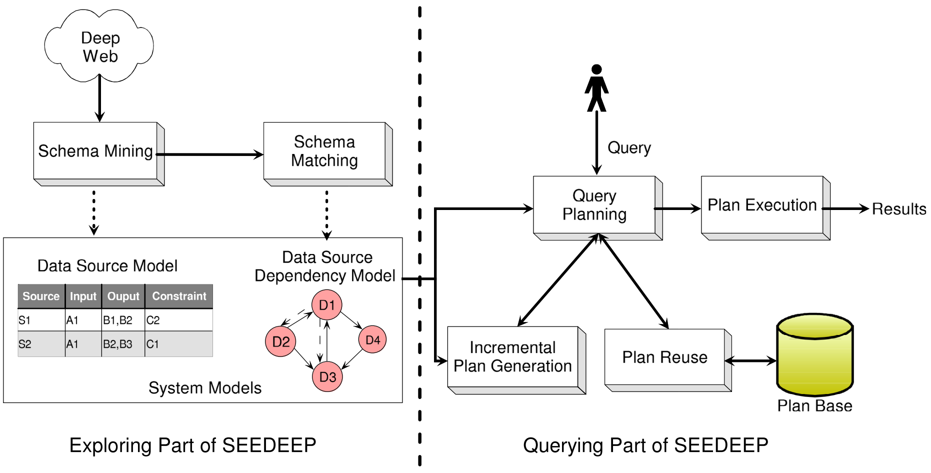


**Fig. 2.** System Infrastructure of SEEDEEP

SEEDEEP is mainly composed of six modules which are *Schema Mining Module*, *Schema Matching Module*, *Query Planning Module*, *Query Reuse Module*, *Incremental Plan Generation Module*, and *Plan Execution Module*.

**Schema Mining (SMi):** SMi is used to automatically mine the complete schemas of deep web data sources and build data source models to describe the data content hidden behind the data sources. The main challenge of mining the complete schema of a deep web data source is that given a particular input term, many data sources will only return a partial set of output schema attributes, i.e., the ones that have non-NULL values for the particular input. SMi utilizes a hybrid algorithm which combines a sampling model and a mixture model to solve this problem.

**Schema Matching (SMa):** When answering a cross-source query, it is important for SEEDEEP to understand the dependence relation among data sources, i.e., the output results of one data source may be the input information of another data source. To construct a data source dependence model, we need to consider the input-output schema matching problem for different data sources. SMa solves this problem using a series of data mining techniques which include automatic discovery of attribute instances and finding matched schema attributes using clustering.

**Query Planning (QP):** The query planning module takes a cross-source keyword query and system models as input, and generates a query plan as shown in Figure 1 to answer the query. In SEEDEEP system, the system models (data source model and dependence model) could be generated using the SMi and SMa modules or our system also allows user provided data source model and dependence model.

**Query Reuse (QR):** The query reuse module takes a *query-plan-driven* caching strategy to generates query plans based on previously cached query plans which are stored in the Plan Base. Different from the widely used *data-driven* caching method based on query containment checking, our strategy could increase the possibility of data reuse even when a new query doesn't seem to be overlapping with a previous query.

**Incremental Plan Generation (IPG):** When a query plan is executed, the remote data sources and communication links are subject to congestion and failures. This can cause significant and unpredictable *delays* or even *unavailability* of sources. The incremental plan generation module gracefully handles such issues using a *data redundancy based incremental query generating* method to generate a partial query plan for the part in the original query plan that became unusable.

**Plan Execution (PE):** The plan execution module achieves three types of parallelism, which are task parallelism, thread parallelism and pipeline parallelism, to accelerate the execution of a query plan.

As shown in Figure 2, SEEDEEP is partitioned into two parts, the exploring part and the querying part. Given a set of deep web data sources, the exploring part, which consists of SMi and SMa, explores the data sources to understand their usages and relationships and build system models. User queries are handled by the querying part of SEEDEEP. The querying part, which contains QP, QR, IPG and PE, finds the most intelligent query plan for a user query based on the system models and executes the plan in the most efficient way.

## 3 System Modules

In this section, we briefly describe the design of each of the six modules in SEEDEEP.

### 3.1 Schema Mining

The schema mining module aims to find the complete output schemas of deep web data sources. Because most deep web data sources only return a partial set of output schema attributes when given a particular input term, the existing label extraction technique which assumes that all output attributes are appearing in the output pages [7] cannot be applied. In our problem scenario, we proposed two schema mining approaches which are *sampling model approach* and *mixture model approach*.

The sampling model approach is based on a distribution model of deep web data source output attributes. We found that a *modest* sized sample of output pages could discover output attributes with relatively high recall. Based on this idea, we proposed a sampling based algorithm to discover output attributes based on a simple random input samples. The size of the sample is estimated by a sample size estimator which could bound the sampling error within a confidence interval. The mixture model approach is based on an observation that there is likely some redundancy across data sources. Thus, attributes could be shared among different data sources. We model a data source as a *probabilistic data source model* that generates output attributes with certain probabilities. Since an attribute could be shared by multiple data sources, we consider the probability of an attribute generated by a data source as being determined by a mixture model composed of the *probabilistic data sources* of similar data sources.

## 3.2 Schema Matching

The schema matching strategy used in SEEDEEP is an instance based method. We focus on finding the input-output attribute matching so as to construct the data source dependence model. The following two methods are proposed to find attribute instance: 1) We take advantage of the informative information provided on the query interfaces of deep web data sources to find attribute instances; 2) Based on the data redundancy across data sources, i.e., the same data can be obtained from multiple data sources [8], we borrow attribute instances from semantically related attributes on the output pages from related data sources. Taking the attribute instances, a clustering algorithm is used to find the semantically matched groups as matched attributes.

## 3.3 Query Planning

Currently, our query planning module is able to handle with two types of cross-source queries, which are 1) *Entity-Attribute Query*, where a user may submit an entity name and one or more attributes, and would like to search based on attributes of interest for the entity, and 2) *Entity-Entity Relationship Query*, where a user submits multiple entity names from a domain, and wants to know possible relationships among these names.

The query planning problem in SEEDEEP can be formulated as a NP-hard problem which is to find a minimal subgraph covering all query terms from the data source dependence graph. Our problem is clearly distinct from the existing query planning problem over web services [9] or relational databases [10,11,12] where a query plan must be a *tree*. We proposed a heuristic bidirectional query planning algorithm to solve our query planning problem. We have a set of *starting nodes* and a set of *target nodes*. A query plan ultimately connects a subset of target nodes with a subset of starting nodes, such that all query terms are covered. We perform backward exploration from the target nodes to connect them with starting nodes, and meanwhile we also do forward exploration from the starting nodes. The heuristic we used to find the minimal subgraph is that the edges that will allow a shorter distance to connect nodes which cover query terms are preferred. A domain ontology and a list of ranking heuristics are also developed to support the query planning module.

## 3.4 Query Reuse

Our query reuse strategy is driven by the following observations. First, there is data redundancy across deep web data sources. Second, deep web data source returns query answers in an *All-In-One* fashion, i.e., values of all the attributes in the source's output schema are returned, irrespective of the specific attributes requested in the query. The above specific features of deep web data sources motivate us to develop a novel *query-plan-driven* caching method.

Given a new query $NQ$, QR module first finds a list of previous queries, denoted as $PQs = \{PQ_1, \ldots, PQ_n\}$, which are similar to $NQ$ based on a similarity metric. Second, from each $PQ_i$, a $\Psi$ selection algorithm selects a query sub-plan $SubP_i$ such that among all valid sub-plans of $PQ_i$, $SubP_i$ *maximally* covers $NQ$ and has the *smallest* size. The query plan for $NQ$ is generated using the list of $SubP_i$ based on a modified query planning algorithm. The modified planning algorithm gives priority to the sub-plans selected by the $\Psi$ algorithm and reduces to the original bidirectional algorithm when there is no $\Psi$ selected sub-plans can be reused.

## 3.5 Incremental Plan Generation

For a query $Q$, an original query plan $P$ is generated. During the execution of plan $P$, a data source $D$ in $P$ experiences unexpected delays or becomes unavailable. We want to find a *maximal inaccessible subplan* rooted at data source $D$, denoted as $MaxSubP$. Then, we want to take advantage of the data redundancy between data sources in $MaxSubP$ and other deep web data sources not in the original plan to incrementally generate a new partial query plan to replace $MaxSubP$.

In our incremental plan generation algorithm, we considered four types of *query plan fixability*, which are *fully fixable*, *partial fixable*, *cascading fully fixable*, and *cascading partial fixable*. For each type of *plan fixability*, we give an algorithm to find the *maximal inaccessible subplan* $MaxSubP$ and generate a new query to replace $MaxSubP$.

## 3.6 Plan Execution

Our plan execution module explores three types of parallelism as follows to accelerate the execution of a query plan. 1) Task Parallelism. Data sources without inter-dependence can be queried in parallel; 2) Thread Parallelism. For a particular data source, multiple input instances can be submitted to the data source in parallel. This is because most deep web data sources support multiple queries simultaneously; 3) Pipeline Parallelism. The output of a data source can be processed by its child(ren) data source(s), while the data source can process new input queries. The detail of the plan execution module can be found in [13].

## 4 System Demonstration and Evaluation

In this section, we describe the current implementation of SEEDEEP and show a case study of SEEDEEP.

## 4.1 System Implementation

Currently, SEEDEEP integrates 16 deep web data sources in the biological domain, which includes `dbSNP`[1], `Entrez Gene`[1], `Protein`[1], `BLAST`[1], `SNP500`[2], `Seattle`[3], `SIFT`[4], `BIND`[5], `Human Protein`[6], `HGNC`[7], `Mouse SNP`[8], `ALFRED`[9], `JSNP`[10],

[1] http://www.ncbi.nlm.nih.gov
[2] http://snp500cancer.nci.nih.gov/home_1.cfm
[3] http://pga.gs.washington.edu/
[4] http://blocks.fhcrc.org/sift/SIFT.html
[5] http://www.bind.ca
[6] www.hprd.org
[7] www.genenames.org
[8] http://mousesnp.roche.com/
[9] http://alfred.med.yale.edu/alfred/
[10] http://snp.ims.u-tokyo.ac.jp/search.html#locus

MGIGene[11], MGISNP[11], and KEGG[12]. The exploring part of the system (SMi and SMa) mines data source schemas and finds attribute matchings. Then system models are constructed for the querying part of SEEDEEP. The system models capture the meta data content of the above 16 data sources and their dependencies. In the querying part, SEEDEEP provides an interface by which users can specify a cross-source keyword query. SEEDEEP uses Apache Tomcat 6.x to support a web server. After a query plan is executed, all results are returned in the form of HTML files to the user. We use Oracle 10 database to implement the plan base in the query reuse module.

## 4.2 Case Study

**Exploring Part of SEEDEEP:** For illustrating the performance of the schema mining and matching module of SEEDEEP, we consider the data source Gene. We first conduct a pilot study to show the characteristics of this data source. For Gene which has a total number of 59 output attributes in its query schema, we randomly create 50 input cases, and analyze the corresponding 50 output pages. We count the number of distinct output attributes. We found that none of the output pages covers the entire output schema, the maximal coverage we obtained is about 80%, and on the average, each output page only covers about 60% of the complete output schema. This pilot study shows that mining query schema of scientific deep web data sources is challenging. The schema mining module yields very good performance on mining the query schema of data sources. For Gene, using only 10 input samples, we obtained a schema attribute recall about 95%. The schema matching module can also correctly identify all dependence relationship between Gene and other data sources integrated. Overall, the schema matching module achieves an average recall of 91% and precision of 96% for all data sources.

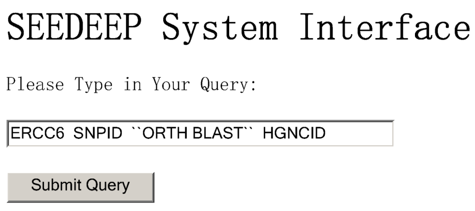
**Querying Part of SEEDEEP:** We use the query Q1={ERCC6,SNPID,"ORTH BLAST",HGNCID}, which is mentioned in Section 1 in the case study. The input interface of SEEDEEP and part of the output result page of query $Q1$ is shown in Figure 3. The query plan generated for $Q1$ by the query planning module is shown in Figure 1. The results are checked by a collaborating biologist and she found they are correct and sufficient.

To show the effectiveness of the query reuse module, we consider a second query Q2={MIMID_609413,SNPID,"SNP ssID", "Protein ID", "ORTH BLAST"} is issued to SEEDEEP. The query reuse module correctly decides to reuse part of the query plan of $Q1$ to answer $Q2$. The reuse strategy speedups the execution of $Q2$ by a factor of 4. Compared with a pure data driven caching method which only can achieve a speedup ratio of 1.5, our plan-driven-caching strategy is more effective.

During the execution of query plan for Q2, we consider data source dbSNP is unavailable. The incremental plan generation module effectively finds a new partial plan and incrementally updates the query plan of Q2 by replacing data source dbSNP with Seattle (dbSNP and Seattle have overlapping data about SNP). The new query

[11] http://www.informatics.jax.org/javawi2/servlet

[12] http://www.genome.jp/kegg/

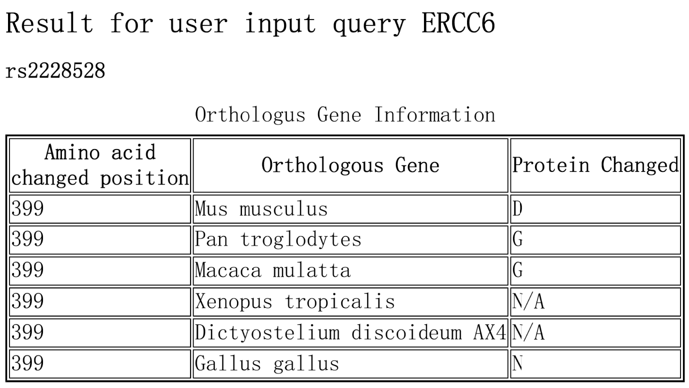


SEEDEEP System Interface

Please Type in Your Query:

ERCC6 SNPID ``ORTH BLAST`` HGNCID

(a)



Result for user input query ERCC6

rs2228528

Orthologus Gene Information

| Amino acid changed position | Orthologous Gene | Protein Changed |
|---|---|---|
| 399 | Mus musculus | D |
| 399 | Pan troglodytes | G |
| 399 | Macaca mulatta | G |
| 399 | Xenopus tropicalis | N/A |
| 399 | Dictyostelium discoideum AX4 | N/A |
| 399 | Gallus gallus | N |

(b)

**Fig. 3.** System Input and Output Example: (a) System Input Interface;(b) Part of System Output Results for Q1

plan saved about 20% of the execution time compared with discarding the original plan and re-generating a new one from scratch.

## 5 Conclusion

In this paper, we have described SEEDEEP, a system for exploring and querying scientific deep web data sources. SEEDEEP comprises two components, which are the exploring and the querying components, with 6 individual modules. SEEDEEP is able to automatically mine query schemas of deep web data sources and find the dependence relation between these data sources. Based on the system models constructed from the exploring part of SEEDEEP, the querying part can generate intelligent plans for cross-source keyword queries. Finally, we also incorporate system-level features like caching and fault tolerance.

## Acknowledgements

This work was supported by NSF grants 0541058 and 0619041. The equipment used for the experiments reported here was purchased under the grant 0403342.

## References

1. He, B., Patel, M., Zhang, Z., Chang, K.C.-C.: Accessing the deep web: A survey. Communications of ACM 50, 94–101 (2007)
2. Babu, P.A., Boddepalli, R., Lakshmi, V.V., Rao, G.N.: Dod: Database of databases–updated molecular biology databases. Silico. Biol. 5 (2005)
3. He, B., Zhang, Z., Chang, K.C.C.: Knocking the door to the deep web: Integrating web query interfaces. In: Proceedings of the 2004 ACM SIGMOD international conference on Management of Data, pp. 913–914 (2004)
4. Chang, K.C.C., Cho, J.: Accessing the web: From search to integration. In: Proceedings of the 2006 ACM SIGMOD international conference on Management of Data, pp. 804–805 (2006)
5. Chang, K., He, B., Zhang, Z.: Toward large scale integration: Building a metaquerier over databases on the web (2005)
6. He, H., Meng, W., Yu, C., Wu, Z.: Automatic integration of web search interfaces with wise_integrator. The international Journal on Very Large Data Bases 12, 256–273 (2004)
7. Zhao, H., Meng, W., Yu, C.: Mining templates from search result records of search engines. In: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 884–891 (2007)
8. Bergman, M.K.: The deep web: Surfacing hidden value. Journal of Electronic Publishing 7 (2001)
9. Kementsietsidis, A., Neven, F., de Craen, D.V., Vansummeren, S.: Scalable multi-query optimization for exploratory queries over federated scientific databases. Proceedings of the VLDB Endowment 1, 16–27 (2008)
10. Hristidis, V., Papakonstantinou, Y.: Discover: Keyword search in relational databases. In: Proceedings of the 28th international conference on Very Large Data Bases, pp. 67–681 (2002)
11. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H.: Bidirectional expansion for keyword search on graph databases. In: Proceedings of the 31st international conference on Very Large Data Bases, pp. 505–516 (2005)
12. Aditya, B., Bhalotia, G., Chakrabarti, S., Hulgeri, A., Nakhe, C., Parag, P., Sudarshan, S.: Banks: Browsing and keyword searching in relational databases. In: Proceedings of the 28th International Conference on Very Large Data Bases, vol. 28, pp. 1083–1086 (2002)
13. Liu, T., Wang, F., Agrawal, G.: Exploiting parallelism to accelerate keyword search on deep-web sources. In: The proceedings of the 2009 DILS workshop (to appear, 2009)

# Expressing OLAP Preferences

Matteo Golfarelli and Stefano Rizzi

DEIS, University of Bologna, Italy
matteo.golfarelli@unibo.it, stefano.rizzi@unibo.it

**Abstract.** Multidimensional databases play a relevant role in statistical and scientific applications, as well as in business intelligence systems. Their users express complex OLAP queries, often returning huge volumes of facts, sometimes providing little or no information. Thus, expressing preferences could be highly valuable in this domain. The OLAP domain is representative of an unexplored class of preference queries, characterized by three peculiarities: preferences can be expressed on both numerical and categorical domains; they can also be expressed on the aggregation level of facts; the space on which preferences are expressed includes both elemental and aggregated facts. In this paper we propose a preference algebra for OLAP, that takes into account the three peculiarities above.

**Keywords:** OLAP, database preferences, multidimensional databases.

## 1 Introduction and Motivation

Personalizing e-services by allowing users to express preferences is becoming more and more common. When querying, expressing preferences is a natural way to avoid empty results on the one hand, information flooding on the other.

Though a lot of research has been carried out during the last few years on database preferences (e.g., [1,2]), the problem of developing a theory of preferences for multidimensional databases has been mostly neglected so far. We argue that expressing preferences could be valuable in this domain because:

- Preferences enable users to focus on the most interesting data. This is particularly beneficial in the OLAP context, since multidimensional databases typically store a huge amount of data. Besides, OLAP queries have a complex structure. An OLAP query may easily return huge volumes of data, or it may return little or no information as well. The data ranking entailed by preferences allows users to cope with both these problems.
- During an OLAP session, the user may not exactly know what she is looking for. The reasons behind a specific phenomenon or trend may be hidden, and finding those reasons by manually applying different combinations of OLAP operators may be very frustrating. Preferences enable users to specify the pattern she is searching for. Since preferences express soft constraints, the most similar data will be returned when no data exactly match that pattern. From this point of view, preference queries can be regarded as a basic OLAM (OnLine Analytical Mining) technique [3].

– Scientific data are often distributed across separate databases. In the business domain, federated data warehouse architectures are seen as an effective shortcut to integration. In these cases, schema heterogeneity may prevent from expressing distributed queries. Conversely, a schema preference can produce meaningful results even when a common schema is not defined.

It is well-known that aggregation plays an essential role in OLAP queries, since it enables decision-makers to get valuable, summary information out of the huge quantity of detailed data available in multidimensional databases. OLAP queries do not only formulate selections and projections on attributes and measures, they also specify on what hierarchy attributes data are to be aggregated (*group-by set*). The aggregation level has a strong impact on the size of the result returned to the user, and its inappropriate setting may end in either obtaining very coarse, useless information or being flooded by tons of detailed data, which is particularly critical when working with devices with small bandwidth and limited visualization capabilities. For this reason we argue that, in the OLAP domain, users may wish to express their preferences on group-by sets too, for instance by stating that monthly data are preferred to yearly and daily data.

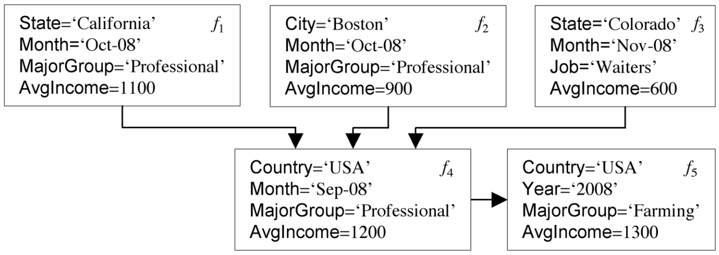


**Fig. 1.** Sample census facts and preference relationships between them

*Example 1.* IPUMS is a public database storing census microdata for social and economic research [4]. An analyst may wish to understand the reasons behind a decrease in the average income of US citizens. She suspects that this is a state-scale phenomenon mainly due to a decrease in the income of professionals. So she expresses a query with a preference on data of professionals, grouped by state, with low income (i.e. lower than \$ 1000). To evaluate this preference, it is necessary to compare data characterized by different group-by sets and different values for attributes and measures. If the analyst is right, this query will return only the facts, aggregated by states, where professionals have an average income lower than \$ 1000. In Figure 1, instead, we assume that the analyst's hypothesis is false and three relevant situations are pointed out. Fact $f_1$ shows that the lowest monthly income of professionals at the state-scale is in California, and it not that low. Fact $f_2$ shows significantly low income for professionals in the city of Boston. Fact $f_3$ shows that, in Colorado, waiters have very low income. Fact $f_4$ is worse than the previous three facts because it yields higher income, is not aggregated by state, and is not related to professionals. Finally, $f_5$ is worse than $f_4$ because it yields an even higher income.

From Example 1, it is apparent that the OLAP domain is representative of an unexplored class of preference queries, characterized by three peculiarities:

– Preferences can be expressed on both attributes and measures, that respectively have categorical and numerical domains. This makes the existing approaches, that are mainly geared to handling either only categorical or numerical preferences, ineffective.
– Preferences can also be formulated on the aggregation level of data, which comes down to expressing preferences on schema rather than on instances. To the best of our knowledge, no approach includes this feature.
– The space on which preferences are declared includes both elemental and aggregated facts. In relational OLAP implementations, materializing all facts on *all* possible group-by sets is highly undesirable.

In this paper we present an approach for dealing with OLAP preferences. Specifically, we propose an algebra for expressing complex OLAP preferences including a set of base preference constructors on attributes, measures and hierarchies, composed by the Pareto operator. The most original of the domain-dependent aspects of our algebra is the possibility of declaring preferences on group-by sets, which is done by recognizing that preferences on the space of hierarchy attributes induce preferences on the space of facts.

## 2 Background Definitions and Working Example

In this section we introduce a basic formal setting to manipulate multidimensional data. To keep the formalism simpler, and without actually restricting the validity of our approach, we will consider hierarchies without branches, i.e., consisting of chains of attributes.

**Definition 1 (Multidimensional Schema).** *A* multidimensional schema *(or, briefly, a* schema*) is a triple $\mathcal{M} = \langle A, H, M \rangle$ where:*

– *$A = \{a_1, \dots a_p\}$ is a finite set of* attributes*, each defined over a categorical domain $Dom(a_k)$;*
– *$H = \{h_1, \dots, h_n\}$ is a finite set of* hierarchies*, each characterized by (1) a subset $Attr(h_i) \subseteq A$ of attributes (such that the $Attr(h_i)$'s for $i = 1, \dots, n$ define a partition of $A$); (2) a* roll-up *total order $\succeq_{h_i}$ over $Attr(h_i)$; and (3) a family of roll-up functions including a function $RollUp_{a_j}^{a_k} : Dom(a_k) \rightarrow Dom(a_j)$ for each pair of attributes $a_k$ and $a_j$ such that $a_k \succeq_{h_i} a_j$;*
– *a finite set of* measures *$M = \{m_1, \dots, m_l\}$, each defined over a numerical domain $Dom(m_i)$.*

For each hierarchy $h_i$, the top attribute of the order is denoted by $DIM_i$, and determines the finest aggregation level of the hierarchy. Conversely, the bottom attribute is denoted by $ALL_i$ and determines the coarsest aggregation level.

**Definition 2 (Group-by Set).** *Given schema $\mathcal{M} = \langle A, H, M \rangle$, let $Dom(H) = Attr(h_1) \times \ldots \times Attr(h_n)$; each $G \in Dom(H)$ is called a* group-by set *of $\mathcal{M}$. Let $G = \langle a_{k_1}, \ldots, a_{k_n} \rangle$ and $Dom(G) = Dom(a_{k_1}) \times \ldots \times Dom(a_{k_n})$; each $g \in Dom(G)$ is called a* coordinate *of $G$. We denote with $G.h_i = a_{k_i}$ the attribute of $h_i$ included in $G$.*

Let $\succeq_H$ denote the product order of the roll-up orders over the hierarchies in $H$. Then, $(Dom(H), \succeq_H)$ is a lattice, that we will call *group-by lattice*, whose top and bottom elements are $G^\top = \langle DIM_1, \ldots, DIM_n \rangle$ and $G^\perp = \langle ALL_1, \ldots, ALL_n \rangle$, respectively.

*Example 2.* Our working schema is CENSUS, that includes hierarchies RES (for “Residence”), OCC (for “Occupation”), and TIME, and measure AvgIncome. The roll-up orders are as follows:

$$\mathsf{City} \succeq_{\mathsf{RES}} \mathsf{State} \succeq_{\mathsf{RES}} \mathsf{Country} \succeq_{\mathsf{RES}} \mathsf{AllCountries}$$
$$\mathsf{Job} \succeq_{\mathsf{OCC}} \mathsf{MinorGroup} \succeq_{\mathsf{OCC}} \mathsf{MajorGroup} \succeq_{\mathsf{OCC}} \mathsf{AllGroups}$$
$$\mathsf{Month} \succeq_{\mathsf{TIME}} \mathsf{Quarter} \succeq_{\mathsf{TIME}} \mathsf{Year} \succeq_{\mathsf{TIME}} \mathsf{AllYears}$$

For instance, it is $RollUp^{\mathsf{City}}_{\mathsf{State}}(\text{'Miami'}) = \text{'Florida'}$. Some examples of group-by sets are $G^\top = \langle \mathsf{City}, \mathsf{Job}, \mathsf{Month} \rangle$, $G_1 = \langle \mathsf{Country}, \mathsf{Job}, \mathsf{Month} \rangle$, $G_2 = \langle \mathsf{State}, \mathsf{AllGroups}, \mathsf{Quarter} \rangle$, and $G^\perp = \langle \mathsf{AllCountries}, \mathsf{AllGroups}, \mathsf{AllYears} \rangle$. It is $G^\top \succeq_H G_1 \succeq_H G^\perp$, $G^\top \succeq_H G_2 \succeq_H G^\perp$, while $G_1$ and $G_2$ are incomparable according to $\succeq_H$. A coordinate of $G_1$ is $\langle \text{'USA'}, \text{'Dentist'}, \text{'Oct-08'} \rangle$.

A schema is populated with facts. A fact is characterized by a group-by set $G$ that defines its aggregation level, by a coordinate of $G$, and by a value for each measure. While the facts at the top group-by set of the lattice (*primary facts*) are those storing elemental data, those at the other group-by sets store summarized information.

**Definition 3 (Fact).** *Given schema $\mathcal{M} = \langle A, H, M \rangle$ and a group-by set $G \in Dom(H)$, a fact at $G$ is a triple $f = \langle G, g, v \rangle$, where $g \in Dom(G)$ and $v \in Dom(M) = Dom(m_1) \times \ldots \times Dom(m_l)$. The space of all facts of $\mathcal{M}$ is $\mathcal{F}_\mathcal{M} = \bigcup_{G \in Dom(H)} (\{G\} \times Dom(G) \times Dom(M))$.*

*Example 3.* An example of fact of CENSUS is $f = \langle G_1, \langle \text{'USA'}, \text{'Dentist'}, \text{'Oct-08'} \rangle, 4000 \rangle$.

Finally, an instance of schema $\mathcal{M}$ is a *datacube* and is defined as a set of facts $C \subset \mathcal{F}_\mathcal{M}$. Intuitively, $C$ includes a set of primary facts at the top group-by set of $Dom(H)$, united with all the other facts determined by aggregating primary facts at all the other group-by sets in $Dom(H)$.

## 3 Preferences on Facts

Before we start to deal with preferences, we briefly recall that a *strict partial order* (s.p.o.) on a given set $S$ is an irreflexive and transitive (thus asymmetric) binary relation on the elements of $S$. A negatively transitive s.p.o. is also called a *weak order* (w.o.). A w.o. on $S$ partitions $S$ into $n$ (disjoint) levels such that the levels are totally ordered and each level determines an SV-relation on the w.o. itself.

In relational databases, a preference is commonly defined as a s.p.o. over the set of possible tuples. Here, we define a preference as a s.p.o. on the space of *all facts at all group-by sets*, which implies that a preference may involve two facts defined at different group-by sets.

**Definition 4 (Preference).** *Given schema $\mathcal{M}$, a* preference *$P$ on $\mathcal{M}$ is a couple $(<_P, \cong_P)$ where $<_P \subseteq \mathcal{F}_\mathcal{M} \times \mathcal{F}_\mathcal{M}$ is a s.p.o. and $\cong_P \subseteq \mathcal{F}_\mathcal{M} \times \mathcal{F}_\mathcal{M}$ is an SV-relation on $<_P$.*

The semantics of $f_1 <_P f_2$ is that $f_2$ is preferred to $f_1$; the semantics of $f_1 \cong_P f_2$ is that $f_2$ is equivalent (or *substitutable*) to $f_1$ [5].

In our approach, complex preferences on the space of facts are inductively engineered by applying a set of base preference constructors and an operator for preference composition. In particular, a preference is defined by a *preference expression* $q$ ruled by the following grammar:

$$< expr >::= < baseConstr > \mid < expr > \otimes < baseConstr >$$
$$< baseConstr >::= \mathsf{POS} | \mathsf{NEG} | \mathsf{BETWEEN} | \mathsf{LOWEST} | \mathsf{HIGHEST} | \mathsf{CONTAIN} | \mathsf{NEAR} | \mathsf{COARSEST} | \mathsf{FINEST}$$

where base preference constructors operate either on attributes, measures, or hierarchies. Adopting the SV-semantics allows for closing the set of composition operators on the set of preferences, thus obtaining an algebra [5].

In the next subsections we will introduce the set of base preference constructors and the composition operator we provide. For simplicity, in this work base preferences are defined over single attributes, measures and hierarchies; the extension to multiple attributes, measures and hierarchies is straightforward and smoothly supported by our approach. Besides, for space reasons, the formalization of some base constructors will be omitted.

## 3.1 Base Preferences on Attributes

While in the relational case each tuple is characterized by the same attributes, the attributes that characterize a fact depend on its group-by set. For instance, a fact reporting the average income for the California state does not explicitly provide values for City and Country. On the other hand, hierarchies allow for relating values of attributes belonging to the same hierarchy by means of roll-up functions. In order to avoid introducing an undesired relationship between preferences on attributes and preferences on hierarchies, in this work we use roll-up functions to propagate preferences expressed on attributes along the *whole* hierarchy, as explained in the following.

Given fact $f$ and hierarchy $h$, let $\bar{a}$ be the attribute of $h$ included in the group-by set of $f$. Then, let $\bar{c}$ be the (categorical) value assumed by $\bar{a}$ in the coordinate of $f$. Given any attribute $a \in Attr(h)$, we denote with $f.a \in 2^{Dom(a)}$ the value(s) assumed in $f$ by $a$, defined as follows:

$$f.a = \begin{cases} \{Rollup_a^{\bar{a}}(\bar{c})\}, \text{ if } \bar{a} \succeq_h a \\ \{c \in Dom(a) | Rollup_{\bar{a}}^{a}(c) = \bar{c}\}, \text{ otherwise} \end{cases}$$

For instance, if $\bar{a}$ = State and $\bar{c}$ = 'California' for fact $f$, it is $f$.City = {'LosAngeles', 'S. Francisco', ...} and $f$.Country = {'USA'}.

Let $c \in Dom(a)$; the base preference constructors we provide for declaring preferences on $a$ are:

– POS($a, c$). Facts whose coordinate on $h$ maps to $c$ are preferred to the others:

$$f_1 <_P f_2 \text{ iff } c \notin f_1.a \ \wedge \ c \in f_2.a$$
$$f_1 \cong_P f_2 \text{ iff } (c \notin f_1.a \ \wedge \ c \notin f_2.a) \vee (c \in f_1.a \ \wedge \ c \in f_2.a)$$

– NEG($a, c$). Facts whose coordinate on $h$ does not map to $c$ are preferred to the others.

It is easy to verify that, for both POS and NEG constructors, $<_P$ is a w.o. and $\cong_P$ is an SV-relation on $<_P$; thus, the result is a preference according to Def. 4.

*Example 4.* POS(Month,'Oct-08') states that the monthly data of October 2008, the daily data for all days of October 2008, and the yearly data for 2008 are preferred to all the other facts.

### 3.2 Base Preferences on Measures

Let $m \in M$ be a measure. Let $v, v_{low}, v_{high} \in Dom(m)$ ($v_{low} \leq v_{high}$); we define

$$\Delta(v, [v_{low}, v_{high}]) = \begin{cases} 0 & \text{if } v \in [v_{low}, v_{high}] \\ v_{low} - v & \text{if } v < v_{low} \\ v - v_{high} & \text{if } v > v_{high} \end{cases}$$

Also, given fact $f$, we denote with $f.m \in Dom(m)$ the (numerical) value assumed in $f$ by $m$. Let $v \in Dom(m)$; the base preference constructors for declaring preferences on measure $m$ are, like in [2]:

– BETWEEN($m, v_{low}, v_{high}$). Facts whose value on $m$ is between $v_{low}$ and $v_{high}$ are preferred; the other facts are ranked according to their distance from the interval:

$$f_1 <_P f_2 \text{ iff } \Delta(f_1.m, [v_{low}, v_{high}]) > \Delta(f_2.m, [v_{low}, v_{high}])$$
$$f_1 \cong_P f_2 \text{ iff } \Delta(f_1.m, [v_{low}, v_{high}]) = \Delta(f_2.m, [v_{low}, v_{high}])$$

– LOWEST($m$), HIGHEST($m$). Facts whose value on $m$ is as low (high) as possible are preferred.

All three constructors return w.o. preferences.

*Example 5.* BETWEEN(AvgIncome, MININC, 1000) states that the facts (aggregated at any group-by set) yielding average incomes lower than 1000 are preferred over the others, that are ranked according to increasing incomes.

## 3.3 Base Preferences on Hierarchies

As stated in the Introduction, one key feature of our approach is the possibility of declaring preferences on the aggregation level of facts, i.e., on their group-by sets. The basic idea is that of defining preferences on the space of hierarchy attributes, to let them induce preferences on the space of facts through a function that maps each fact into its group-by set. In particular, given fact $f$ and hierarchy $h$, let $G(f)$ denote its group-by set and $G(f).h$ denote the attribute of $h$ in $G(f)$.

**Definition 5 (G-order and G-relation).** *Let $<_{P'}$ and $\cong_{P'}$ be, respectively, an order and an equivalence relation on the attributes of $h$, $Attr(h)$. We call* G-order *the order $<_P$ induced on $\mathcal{F}_{\mathcal{M}}$ by $<_{P'}$ as follows: for each $f_1, f_2 \in \mathcal{F}_{\mathcal{M}}$, it is $f_1 <_P f_2$ iff $G(f_1).h <_{P'} G(f_2).h$. We call* G-relation *the equivalence relation $\cong_P$ induced on $\mathcal{F}_{\mathcal{M}}$ by $\cong_{P'}$ as follows: for each $f_1, f_2 \in \mathcal{F}_{\mathcal{M}}$, it is $f_1 \cong_P f_2$ iff $G(f_1).h \cong_{P'} G(f_2).h$.*

Theorem 1 shows that the properties of the relationships on hierarchy attributes are preserved in the relationships induced on facts through $G()$.

**Theorem 1.** *Let $<_{P'}$ and $\cong_{P'}$ be, respectively, an order and an equivalence relation on $Attr(h)$, and let $<_P$ and $\cong_P$ be their G-order and G-relation, respectively. Then, $(<_P, \cong_P)$ is a preference iff $(<_{P'}, \cong_{P'})$ is a preference. Besides, $<_P$ is a w.o. iff $<_{P'}$ is a w.o.*

We now introduce the notion of distance between two attributes in a hierarchy $h$, that is necessary for declaring NEAR preferences:

**Definition 6 (Distance).** *Let $a_1, a_2 \in Attr(h)$. The* distance *between $a_1$ and $a_2$, $Dist(a_1, a_2)$, is the difference between the levels of $a_1$ and $a_2$ within the roll-up total order $\succeq_h$.*

For instance, with reference to the CENSUS schema, it is $Dist(\mathsf{Month}, \mathsf{Year}) = 2$.

Given $a, a_{fine}, a_{coarse} \in Attr(h)$, $a_{fine} \succeq_h a_{coarse}$, let

$$\Delta(a, [a_{fine}, a_{coarse}]) = \begin{cases} 0, \text{ if } a_{fine} \succeq_h a \succeq_h a_{coarse} \\ min\{Dist(a, a_{fine}), Dist(a, a_{coarse})\}, \text{ otherwise} \end{cases}$$

We are now ready to define the following base preference constructors on hierarchies. Let $h \in H$ and $a, a_{fine}, a_{coarse} \in Attr(h)$ $(a_{fine} \succeq_h a_{coarse})$:

- CONTAIN$(h, a)$. The facts whose group-by set includes $a$ are preferred to the others:

  $f_1 <_P f_2$ iff $a \neq G(f_1).h \ \wedge \ a = G(f_2).h$

  $f_1 \cong_P f_2$ iff $(a \neq G(f_1).h \ \wedge \ a \neq G(f_2).h) \vee (a = G(f_1).h \ \wedge \ a = G(f_2).h)$

- NEAR$(h, a_{fine}, a_{coarse})$. The facts whose group-by set along $h$ is between $a_{fine}$ and $a_{coarse}$ are preferred; the other facts are ranked according to the distance of their group-by set along $h$ from the interval:

  $f_1 <_P f_2$ iff $\Delta(G(f_1).h, [a_{fine}, a_{coarse}]) > \Delta(G(f_2).h, [a_{fine}, a_{coarse}])$

  $f_1 \cong_P f_2$ iff $\Delta(G(f_1).h, [a_{fine}, a_{coarse}]) = \Delta(G(f_2).h, [a_{fine}, a_{coarse}])$

– COARSEST($h$). Aggregated facts along $h$ are preferred to detailed ones:

$$f_1 <_P f_2 \text{ iff } G(f_1).h \succeq_h G(f_2).h$$
$$f_1 \cong_P f_2 \text{ iff } G(f_1).h = G(f_2).h$$

– FINEST($h$). Detailed facts along $h$ are preferred to aggregated ones.

All four constructors return w.o. preferences over $Attr(h)$ and, for Theorem 1, w.o. preferences over $\mathcal{F}_{\mathcal{M}}$. FINEST returns the roll-up order on $h$, $\succeq_h$.

*Example 6.* CONTAIN(RES,State) selects a set of preferred group-by sets (those including State combined with any attribute of OCC and TIME). In other terms, it states that the census data aggregated by residence state are preferred to the others, regardless of what their aggregation is on the occupation and time hierarchies. NEAR(TIME,Quarter,Year) states that data aggregated by either quarter or year are preferred to the others. Data aggregated by month and data completely aggregated along the time hierarchy are substitutable. FINEST(TIME) ranks the group-by sets (and their facts) according to the roll-up lattice of TIME.

## 3.4 Preference Composition

The most common operator for preference composition is the Pareto operator:

– $P_1 \otimes P_2$ (Pareto composition). A fact is better than another if it is better according to one preference and better or substitutable according to the other (the composed preferences are considered equally important):

$$\begin{aligned} f_1 <_{P_1 \otimes P_2} f_2 \text{ iff } & (f_1 <_{P_1} f_2 \wedge (f_1 <_{P_2} f_2 \vee f_1 \cong_{P_2} f_2)) \\ & \vee (f_1 <_{P_2} f_2 \wedge (f_1 <_{P_1} f_2 \vee f_1 \cong_{P_1} f_2)) \\ f_1 \cong_{P_1 \otimes P_2} f_2 \text{ iff } & f_1 \cong_{P_1} f_2 \wedge f_1 \cong_{P_2} f_2 \end{aligned}$$

As reported in [5], Pareto composition with SV-semantics preserves s.p.o.'s. Thus, the result of applying this composition operator starting from the base preference constructors defined in this section is still a preference according to Def. 4. Note that Pareto composition is commutative and associative.

*Example 7.* The preference query introduced in Example 1 can be formulated as BETWEEN(AvgIncome,MININC,1000) $\otimes$ CONTAIN(RES,State) $\otimes$ POS(MajorGroup,'Professional').

## 4 Conclusions and Related Works

In this paper we have argued that preferences are a valuable technique for many OLAP applications. On the other hand, ad-hoc base preference constructors are needed to handle the required expressiveness. Thus, we have defined an algebra that allows preferences to be formulated, besides attributes and measures, also on hierarchies, i.e., on the aggregation level of facts.

The literature on preference queries is huge, but only few works may be related to queries involving preferences on schema and aggregated data. An attempt to situate preferences in the context of multidimensional databases is [6], whose focus is to enable efficient computation of Boolean predicates and preference expressions on numerical domains. Preferences on categorical domains are not supported, and there is no mention to the possibility of expressing preferences on aggregation levels. Finally, in [7] preferences are expressed on a hierarchy of concepts, but information is always retrieved at the finest level of detail and preferences cannot be expressed on schema.

To close this section, we briefly discuss the effectiveness of our approach. We start by observing that OLAP preferences play a major role in reducing the effort of decision-makers to find the most interesting information. This effort can be quantitatively estimated by counting the number of OLAP queries necessary to "manually" retrieve the facts that best match the user preferences. Running an OLAP session entails formulating a sequence of queries, each specifying a group-by set, a list of required measures and an optional set of predicates on attributes and measures. To minimize her effort, a decision-maker should run an OLAP session by first formulating queries that may return best-matching facts. For instance, consider the preference in Example 7. To manually obtain the same results, the decision-maker should first formulate all possible queries including State (i.e., 16 queries) in the group-by set and select the facts related to professionals and yielding an average income lower than $ 1000. However, if no facts exactly matching the preference are found, more queries will be required; in the worst case, 256 queries must be formulated, which means retrieving the whole datacube. In presence of complex preferences on measures (such as HIGHEST(AvgIncome) $\otimes$ LOWEST(AvgMortgage)), the decision-maker would also have to analyze the results to check for numerical domination since this cannot be expressed by an OLAP query.

## References

1. Chomicki, J.: Preference formulas in relational queries. ACM Trans. on Database Systems 28(4), 427–466 (2003)
2. Kießling, W.: Foundations of preferences in database systems. In: Proc. VLDB, Hong Kong, China, pp. 311–322 (2002)
3. Han, J.: Towards on-line analytical mining in large databases. ACM SIGMOD Record 27, 97–107 (1998)
4. Minnesota Population Center: Integrated public use microdata series (2008), http://www.ipums.org
5. Kießling, W.: Preference queries with SV-semantics. In: Proc. COMAD, Goa, India, pp. 15–26 (2005)
6. Xin, D., Han, J.: P-cube: Answering preference queries in multi-dimensional space. In: Proc. ICDE, Cancún, México, pp. 1092–1100 (2008)
7. Koutrika, G., Ioannidis, Y.: Answering queries based on preference hierarchies. In: Proc. VLDB, Auckland, New Zealand (2008)

# Energy Smart Management of Scientific Data

Ekow Otoo, Doron Rotem, and Shih-Chiang Tsao

Lawrence Berkeley National Laboratory,
University of California
1 Cyclotron Road
Berkeley, CA 94720

**Abstract.** Scientific data centers comprised of high-powered computing equipment and large capacity disk storage systems consume considerable amount of energy. Dynamic power management techniques (DPM) are commonly used for saving energy in disk systems. These involve powering down disks that exhibit long idle periods and placing them in standby mode. A file request from a disk in standby mode will incur both energy and performance penalties as it takes energy (and time) to spin up the disk before it can serve a file. For this reason, DPM has to make decisions as to when to transition the disk into standby mode such that the energy saved is greater than the energy needed to spin it up again and the performance penalty is tolerable. The length of the idle period until the DPM decides to power down a disk is called idleness threshold.

In this paper, we study both analytically and experimentally dynamic power management techniques that save energy subject to performance constraints on file access costs. Based on observed workloads of scientific applications and disk characteristics, we provide a methodology for determining file assignment to disks and computing idleness thresholds that result in significant improvements to the energy saved by existing DPM solutions while meeting response time constraints. We validate our methods with simulations that use traces taken from scientific applications.

**Keywords:** Disk storage, Power management, File allocation, Scientific workload, Performance guaranttee.

## 1 Introduction

The rapid growth in highly data-intensive scientific research has fueled an explosion in computing facilities and demand for electricity to power them. Several analysts are now predicting that energy costs will eventually outstrip the cost of hardware in data centers [1]. As a result, reducing energy costs at data centers has become the focus of multiple research efforts which are aimed at devising architectural strategies for energy efficient computing systems. Examples of projects that are currently underway include the GreenLight project at UC San Diego, DiskEnergy at Microsoft, GREEN-NET Project in INRIA and the Green Grid Consortium.

There are multiple components that contribute to the power consumption in a data center such as servers, storage, cooling, networks etc. However, recent papers estimate that about 25 -35 percent of the energy consumption at data centers is attributed to disk storage systems [2]. This percentage of disk storage power consumption will continue to increase, as faster and higher capacity disks are deployed with increasing energy costs and also as data intensive applications demand reliable on-line access to data resources.

Reducing the energy consumption of the disk storage system has been addressed in many recent research works. Research efforts are directed at several levels such as *physical device* level , *systems* level and dynamic power management (DPM) algorithms. At the physical device level, disk manufacturers are developing new energy efficient disks [3] and hybrid disks (i.e., disks with integrated flash memory caches). At the system level, a number of integrated storage solutions such as MAID [4], PARAID [5], PERGAMUM [6] and SEA [7] have emerged all of which are based on the general principle of spinning down and spinning up disks. Disks configured either as RAID sets or as independent disks, are configured with idle time-out periods, also called *idleness threshold*, after which they are automatically spun down into a standby mode. A read or write I/O request targeted to a standby disk causes the disk to spin-up in order to service it. This of course comes at the expense of a longer response time to file access requests as well as a penalty in terms of energy costs.

Dynamic power management (DPM) algorithms have been proposed to determine online when the disk should be transitioned to a lower power dissipation state while experiencing an idle period. Analytical solutions to this online problem have been evaluated in terms of their competitive ratio. This ratio is used to compare the energy cost of an online DPM algorithm to the energy cost of an optimal offline solution which knows the arrival sequence of disk access requests in advance. It is well known [8] that for a two state system where the disk can be in either standby or in idle mode there is a tight bound of 2 for the competitive ratio of any deterministic algorithm. This ratio is achieved by setting the idleness threshold, $T_\tau$, to $\frac{\beta}{P_\tau}$ where $\beta$ is the energy penalty (in joules) for having to serve a request while the disk is in standby mode, (i.e., spinning the disk down and then spinning it up in order to serve a request) and $P_\tau$ is the rate of energy consumption of the disk (in watts) in the idle mode. We call this value the *competitive idleness threshold*.

In this paper we focus mainly on read requests, we assume that write requests can be handled efficiently by using any one of the energy-friendly approaches presented in the literature. For example, in [6] it is recommended that files will be written into an already spinning disk if sufficient space is found on it or write it into any other disk (using best-fit or first-fit policy) where sufficient space can be found. The written file may be re-allocated to a better location later during a reorganization process. Another recently proposed strategy for energy saving for writes is called Write Off-Loading [9]. This technique allows write requests on spun-down disks to be temporarily redirected reliably to persistent storage elsewhere in the data center.

## 1.1 Contributions of This Paper

In this paper, we quantify the effects of disk power management on response time based on request workloads and disk characteristics. To the best of our knowledge, with the exception of the work in [10], very little work has been done on modeling and analyzing the effects of power management on the response time for file access requests using realistic workloads and disk characteristics. In addition, the trade-off associated with using more or less disks on power consumption and response times has not been studied.

More specifically, our goal is to produce useful tools that can help in determining when power saving policies should be used at all as well as optimal idleness thresholds and bounds on the number of required disks needed in order to provide response time guarantees. Our main contributions are:

– We develop an analytic model of file requests served by a disk equipped with power saving mechanisms
– Based on this model, we develop a procedure called SmartIdle that computes several important parameters such as request arrival rate ”break-even” point that determines when power saving should be applied, how many disks must be used to support response time constraints, and optimal idleness thresholds
– We validate this procedure by applying it to two real life scientific application traces and show significant improvement over common existing DPM strategies that use *competitive idleness threshold* value to power down the disk.

The remainder of the paper is organized as follows. More details about related relevant work are provided in Section 2. In Section 3 our analytical model is described. In Section 4 we present our procedure for determining system parameters for maximizing energy savings while meeting performance requirements. In Section 5 we present our simulation model and results and in Section 6 an application of our model on two scientific workloads is presented. Finally in Section 7 we present our conclusions and directions for future work.

## 2 Related Work

Conserving energy in large scale computing has been recently explored in [11, 12]. Colarelli and Grunwald [4] proposed *MAID* for near-line access to data in a massively large disk storage environment. They show, using simulation studies, that a MAID system is a viable alternative and capable of considerable energy savings over constantly spinning disks. A related system was implemented and commercialized by COPAN systems [12, 13]. This system, which is intended for a general data center, is not focused on scientific applications and is not adaptively reconfigurable based on workloads.

The theory of Dynamic Power Management of disks has drawn a lot of attention recently from the theoretical computer science community (see [8] for an extensive overview of this work). Most of this work considers a single disk only and attempts to find an optimal idle waiting period (also called idleness threshold time) after which a disk should be moved to a state which consumes less power. More specifically, the problem discussed in these research works is based on the assumption that the disk can be transitioned among $n$ power consumption states where the $i^{th}$ state consumes less power than the $j^{th}$ state for $i < j$. The disk can serve file requests only when it is in the highest power state (the $n^{th}$ state) which is also called the active state. The system must pay a penalty $\beta_i$ if a request arrives when the disk is in the $i^{th}$ state, the penalty is proportional to the power needed to spin up from state $i$ to the active state $n$. The penalty is decreasing with the state number, i.e., $\beta_j < \beta_i, for j > i,\ and\ \beta_n = 0$.

The problem is that of devising online algorithms for selecting optimal threshold times, based on idle periods between request arrivals, to transition the disk from one power state to another. The most common case transitions between two states namely, idle state (full power) and standby or sleep state (zero power). The quality of these algorithms is measured by their competitive ratio which compares their power consumption to that of an optimal offline algorithm that can see the entire request sequence in advance before selecting state transition times. As mentioned before, for a two state system there is a tight bound of 2 for the competitive ratio of any deterministic algorithm.

There are also several results showing that with randomized online algorithms, the best competitive ratio achievable improves to $e/(e-1) \approx 1.58$ [14]. Response time penalty is not considered in these works. Another approach to DPM [15], attempts to learn the request arrival sequence probability based on previous history and then generates a probability-based DPM strategy that minimizes the expected power dissipation. It is known that power management schemes have an effect on the response time of the system. In [16] an upper bound on the additional latency of the sys- tem introduced by power management strategies is established.

More recently, it has been suggested that energy efficiency issues should become a first-class performance goal for query processing in large data base management systems. Several research papers deal with energy efficiency in DBMS using several benchmarks. Examples include the JouleSort [17]and SPECPower benchmarks which measure energy efficiency of entire systems that perform data management tasks.

In [18], the authors develop a power consumption model based on data from the TPC-C benchmark. In [19], the authors provide a framework for trading off performance and power consumption of storage systems based on a graduated, distribution-based QoS model. This work deals with workload profiling partitioning and scheduling to reduce energy consumption. In [20] energy-efficiency optimizations within database systems are discussed. The experiments in [20] use a decision support workload (TPC-H) which scans an entire table and applies a predicate to it. In [21] techniques for reducing power consumption in DBMS are introduced. One such technique, called QED, uses well known query aggregation methods to leverage common components of queries in a workload to reduce accesses to the storage system. The technique involves some performance penalties as it is done by delaying some queries in order to increase such leveraging opportunities. Other energy conservation techniques proposed are addressed in [6,11,5,9,22,10].

## 3 Model

### 3.1 Definitions and Notations

In the section, we apply the M/G/1 queuing model, similar to the approach in [23,7,10], to estimate the power cost and response time for a disk with a specific exponential arrival rate of file access and idleness threshold. Table 1 displays the notations and parameters used in the model. The values for the parameters of disk, e.g. $T_d$, $T_u$, $P_u$, and $P_d$, are given based on the specification in [3].

### 3.2 Power Cost

In this section $E[Y]$ denotes the expected value of the variable $Y$. In the following section we present an anlytical model for estimating the power costs for $D_{PS}$ and $D_{NPS}$ within

**Table 1.** Notations and Disk Parameter values

| ***Name*** | ***Notation*** | ***Default Value*** |
|---|---|---|
| $D_{PS}$ | disk with power-saving mode | |
| $D_{NPS}$ | disk without power-saving mode | |
| $G_{PS}$ | Energy cost in one cycle by $D_{PS}$ (J) | |
| $T_{PS}$ | The length of one cycle for D$_{PS}$ (s) | |
| $P_{PS}$ | Power cost of $D_{PS}$ (W) | $E_{PS}/T_{PS}$ |
| $P_{NPS}$ | Power cost of D$_{NPS}$ (W) | |
| $P$ | $P_{PS}/P_{NPS}$ | |
| $T_\tau$ | Idleness Threshold (s) | $10 \sim 500$ |
| $T_d$ | Time to spin down a disk (s) | 10 |
| $T_u$ | Time to spin up a disk (s) | 15 |
| $P_d$ | Power to spin down a disk (W) | 9.3 |
| $P_u$ | Power to spin up a disk (W) | 24 |
| $P_\tau$ | Power in idle mode (W) | 9.3 |
| $P_a$ | Power in active mode (W) | 13.0 |
| $P_{sby}$ | Power in standby mode (W) | 0.8 |
| $G_{du}$ | Energy to spin down and up a disk (J) | $P_d \times T_d + P_u \times T_u$ |
| $T_a$ | Length of a busy period entered from an idle state (s) | |
| $f_{T_a}(x)$ | Length of a busy period entered from an $x$-second warm-up state, consisting of partial $T_d$ and the whole $T_u$ (s) | |
| $\lambda$ | arrival rate of file access (1/s) | $0.1 \sim 0.001$ |
| $\rho$ | traffic intensity for the disk | $\lambda \times E[S]$ |
| $E[S]$ | Mean service time of a file (s) | 7.56s |
| $E[S^2]$ | | 178.05s |

one cycle of power mode transitions of a disk, where one cycle represents the time from the end of a busy period to that of the next busy period. Since one cycle of $D_{NPS}$ must consist of an idle period and a busy period, we can express the mean value of $P_{NPS}$ as

$$E[P_{NPS}] = \frac{P_\tau \times 1/\lambda + P_a T_a}{1/\lambda + T_a}.$$

However, for $D_{PS}$ there are three different patterns in one cycle, as shown in Figure 1 where $t$ represents the time from the end of the last busy period to the arrival of the next request. Recall that under an M/G/1 model, if the arrival rate is $\lambda$, the time between two busy periods, $t$, is an exponential distribution with mean = $1/\lambda$. $T_a$ denotes the length of a busy period entered from an idle state. We know that the mean of $T_a$ under an M/G/1 model can be expressed as

$$E[T_a] = E[S]/(1-\rho);$$

where $S$ denotes the service time and $\rho$ is the traffic intensity. Also, $f_{T_a}(X)$ represents the length of a busy period entered from an $x$-second spinning-up state, consisting of

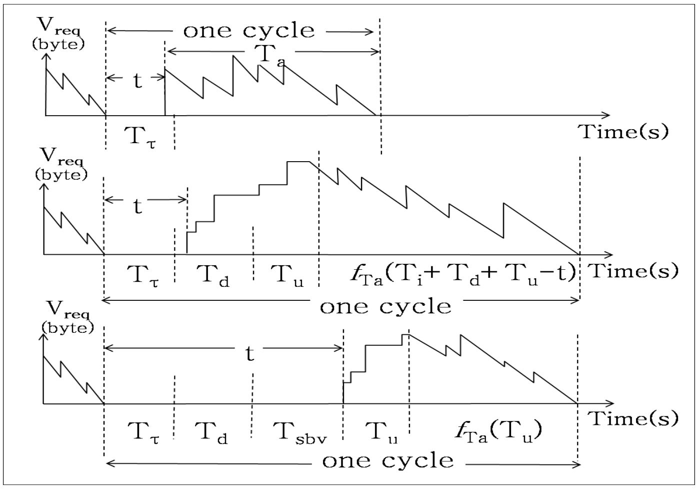


**Fig. 1.** The three possible patterns in a cycle for DPS. The vertical axis, $V_{req}$ represents the total volume, in bytes, of unserviced requests.

partial $T_d$ and the whole of $T_u$. Under the M/G/1 model with setup time, the mean of $f_{T_a}(X)$ can be written as

$$E[f_{T_a}(x)] = (1+\lambda \times x)E[T_a];$$

where $x$ is the setup time [2], i.e., equivalent to the spin-up time in this work.

Since each case has different occurrence probability in a period of one cycle, $E[P_{PS}]$ can be expressed as

$$E[P_{PS}] = \frac{E[G_{PS}]}{E[T_{PS}]} = \frac{\sum_{i=1}^{3} E[G_{PS}^i]P^i}{\sum_{i=1}^{3} E[T_{PS}^i]P^i};$$

where $E[G_{PS}^i]$, $E[T_{PS}^i]$, and $P^i$ are the mean energy cost, mean time period and probability of a request arrival in Case $i$, respectively. The following three Cases occur depending on the arrival of a request. The details of power cost and response times calculations are given in Appendix A.

***Case 1,* $t < T_\tau$:** This case indicates that a request arrives while the disk is idle but before the idle period reaches the Idleness threshold value for it to begin spinning-down.

***Case 2,* $T_\tau \le t < T_\tau + T_d$:** Here an request arrives when the disk has been long enough past its idleness threshold; it is in the process of spinning down but has not completely spun-down.

***Case 3,* $(T_\tau + T_d) \le t$:** In this case the request arrives after the disk has completely spun-down.

The mean response times can similarly be estimated for disks operating in power saving mode $D_{PS}$. We can calculate the mean sojourn time $\theta$ of a request, i.e., its response time, by calculating the mean $E[\theta]$ for each case. The details of these calculations are given in Appendix A.

### 3.3 Numerical Results

In this section, we illustrate our methods using the disks and workload characteristics in Tables 1 and 2. Similar figures can be obtained using the analytical model developed. Figure 2 plots the relationship between $E[P_{PS}]/E[P_{NPS}]$ and $\lambda$ for $T_\tau = 0, 10, 53, \text{and} 160$. When $\lambda < 0.029$, $P_{PS}/P_{NPS}$ would be smaller than 1, i.e., the power-saving mechanism is efficient since it is below the threshold of $\lambda$. Figure 3 plots the corresponding values for response times including the case for $T_\tau = \infty$. We note that $T_\tau = 53$ is the *competitive idleness threshold* in our case obtained by $G_{du}/(P_\tau - P_{sby})$.

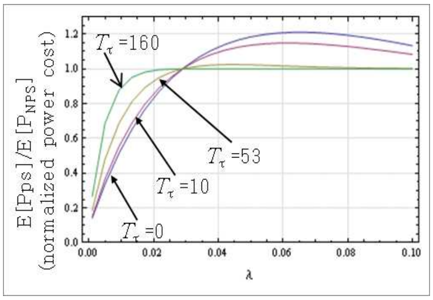


**Fig. 2.** Relationship between the ratio of $E[P_{PS}]/E[P_{NPS}]$ and arrival rate $\lambda$ for $T_\tau = 0, 10, 53$ and 160s

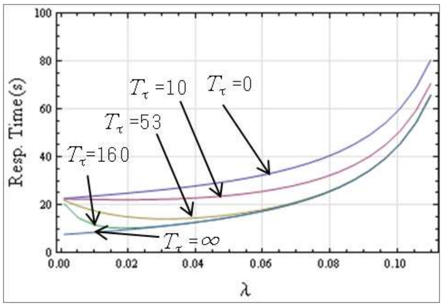


**Fig. 3.** Graphs of response time vs. arrival rate $\lambda$, for $T_\tau = 0, 10, 53, 160$ and $\infty$ sec.

From Figure 2, we note that when $\lambda > 0.029$, the $D_{PS}$ disk have a normalized power cost larger than 1. That is, when the arrival rate of files in a power-saving ($D_{PS}$) disk is larger than 0.029, then its power saving features should be turned off to avoid incurring more power cost than a non-power-saving (NPS) disk.

## 4 Procedure for Selecting Parameters of Disk Storage Configuration

### 4.1 Procedure

Figure 3 describes the relationship between the arrival rate of requests to one disk and their corresponding expected response time. In Figure 4 the curves $\theta_{PS}(\lambda, 0)$ and $\theta_{PS}(\lambda, \infty)$ represent the mean response times of disks hit with request arrivals at rate $\lambda$ for $T_\tau = 0$ and $T_\tau = \infty$ respectively. The entire space covered by Figure 4 is divided into 5 areas based on the following rules. Let $\theta'$ denote the required constraint, by the user, on the response time. We will use this figure to describe our procedure called *SmartIdle* presented as Procedure 1. The procedure will determine the necessary number of active disks and the idleness threshold for these disks. Let $R$ denote the total arrival rate of requests to the system, and suppose the minimum number of disks required to hold the entire set of files is $N$. The procedure first computes the arrival rate for a single disk, $\lambda = R/N$. Given a point $X$ with coordinates $\langle \lambda, \theta' \rangle$ which represents a combination of

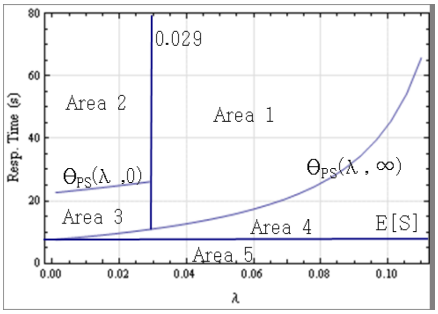


**Fig. 4.** Five areas of the procedure

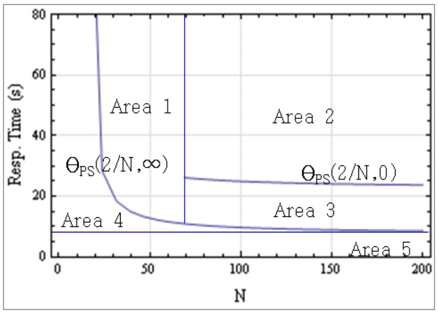


**Fig. 5.** An alternative plot of 4, when $R = 2$ per sec

request arrival rate and the required response time in Figure 4, we will show how to calculate the actual number of active disks and the idleness threshold based on the area that contains this point. First, if $\lambda \geq 0.029$, since power-saving mechanism is inefficient according to Figure 2, the procedure will suggest spinning disks for the whole time, i.e., set $i$ the idleness threshold to $\infty$. Thus, for any $\theta'$ larger than $\theta_{PS}(\lambda,\infty)$ and $\lambda > 0.029$ (i.e., Area 1), we know $N$ disks are enough to meet such constraints because they can offer $\theta_{PS}(\lambda,\infty)$ of response time, which is smaller than $\theta'$. Note that, in this case using less than $N$ disks, in order to save power is not feasible because $N$, is the minimum of disks necessary for storing the entire data.

Second, if $\lambda < 0.029$ but $\theta' > \theta_{PS}(\lambda,0)$ i.e., the point $X$ falls in Area 2 of Figure 4. We know that such a constraint can be satisfied even when the disk is spun down if there are no requests pending for service. Thus, $N$ disks are enough and their idleness threshold will be set to 0 to save the most power. Using more than $N$ disks is not useful because the constraint on response time is always satisfied. In the case that $\theta_{PS}(\lambda,\infty) < \theta' < \theta_{PS}(\lambda,0)$, (i.e. $X$ lies in Area 3), it is necessary to carefully get an idleness threshold $T_\tau$ that satisfies $\theta_{PS}(R/N, T_\tau) = \theta'$.

Third, if the given $\theta' < \theta_{PS}(\lambda,\infty)$, (i.e., Area 4), we have that the arrival rate is too high for each disk to finish serving a file within $\theta'$ even when the power-saving mechanism is disabled to avoid the additional delay of spinning-up and spinning-down disks. In this case, more than $N$ disks are needed to obtain a $\lambda = R/N$ to be sufficiently low to satisfy $\theta_{PS}(\lambda,\infty) = \theta'$. Finally, if the point $X$ falls in Area 5, the given $\theta'$ is not feasible to be satisfied because it is smaller than the service time. The *Procedure* 1, describes the process of estimating the required number of disks and expected response times for configuring a system of disks given a usage workload and specific disks characteristics.

## 4.2 Illustration

The following gives an example on how to use the procedure. Assuming the arrival rate of requests is fixed at 2 per sec, we can redraw Figure 4 to show the relationship between $N$ and the response time. This is now shown in Figure 5. Observe that Figure 5 can be considered a Y-Axis mirror image of Figure 4. For $N = 50$, if a response time within 20 seconds is desired, then Area 1 of Figure 5 should be used. This implies that the files should be stored on 50 disks which are kept constantly spinning. However, if an average response time less than 10 seconds is desired, then $X$ falls within Area 4. In this

```
Procedure SmartIdle($R, N, \theta', \lambda, i$)
    Input:
      $R$ : The total arrival rate of files to the system.
      $N$ : The minimum number of disks to store the data.
      $\theta'$ : The constraint on the response time.
      $\lambda$ : Arrival rate of file requests to a disk.
      $i$ : The idleness threshold.
    Output:
      P : The expected mean power cost
      $\theta$ : The expected mean response time
    $\theta_{PS}(\lambda, i)$ defines a function for the mean response time of disk hit with
    request arrival rate $\lambda$ and an idleness threshold of $i$. ;
    Set $\lambda \Leftarrow R/N$; $X \Leftarrow$ coordinates$\langle\lambda, \theta'\rangle$ ;
    switch Area that X lies do
        case Area 1
            Pack files into $N$ disks that are never spun-down;
            Set $P \Leftarrow N * P_{NPS}(a)$;  $\theta \Leftarrow \theta_{NPS}(\lambda)$  alternatively $\theta \Leftarrow \theta_{PS}(\lambda, \infty)$ ;
            break ;
        case Area 2
            Pack files into $N$ disks ;
            Set $i \Leftarrow 0$; $P \Leftarrow N * P_{PS}(\lambda, 0)$; $\theta \Leftarrow \theta_{PS}(\lambda, 0)$ ;
        case Area 3
            Set idleness threshold $i$ that satisfies $\theta_{PS}(R/N, i) = \theta'$ ;
            Pack files into $N$ disks ;
            Set $P \Leftarrow N * P_{PS}(\lambda, i)$; $\theta \Leftarrow \theta_{PS}(\lambda, i)$ ;
        case Area 4
            Left shift the point $X$ until it intersects the curve ;
            Pack files into $M$ disks such that $M$ satisfies $\theta_{PS}(R/N, \infty) = \theta'$ ;
            // These are disks that are never spun-down
            Set $P \Leftarrow M \times P_{NPS}(R/M)$; $\theta \Leftarrow \theta_{NPS}(R/M)$ ;
        case Area 5
            No solution can satisfy the specified constraints ;
```

case we compute $N$ such that $\theta_{PS}(2/N, \infty) = 10$. The solution is given by $N = 80$. This means that files should be distributed over 80 instead of 50 disks that are constantly spinning.

Next, if $N = 100$ and the response time constraint is set at 15 sec, then the point falls in Area 3. To get the possible idleness thresholds, we examine, from Figure 6 those curves that cross the line of $\theta = 15$ when $N \geq 100$. Such a position is marked by the symbol $X$, where the two curves $\theta_{PS}(2/N, 53)$ and $\theta_{PS}(2/N, 80)$ cross the line. So we still use 100 disks to pack the files and set the idleness threshold at 53 seconds, i.e., the smaller of 53 and 80. This saves more power while meeting the 15-sec constraint. However, if the response time constraint is 40-sec, the point falls in Area 2 In this case we distribute files into the 100 disks again, but set the idleness threshold to zero. This

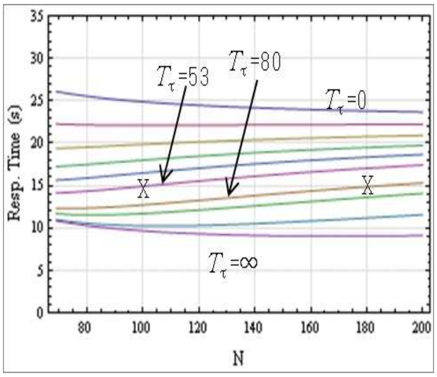


**Fig. 6.** A enlarged area 3 for multiple curves of $\theta_{PS}(2/N,1)$

means a disk is spun-down as soon as no requests are pending for service. This not only saves power but also provides 25 seconds of average response time.

## 5 The Simulation

We developed a simulation model to examine the model proposed in Section 3 and compared this with the SmartIdle procedure described in Section 4. The simulation environment was developed and tested using SimPy [24], as illustrated in Figure 7. The environment consists of a workload generator, a file dispatcher, and a group of hard disks.

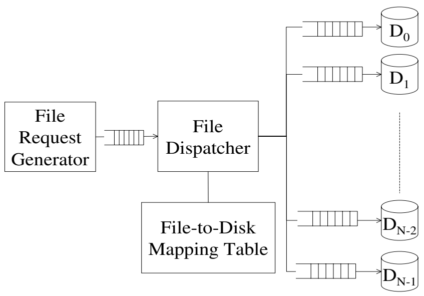


**Fig. 7.** The configuration of disks in the simulation

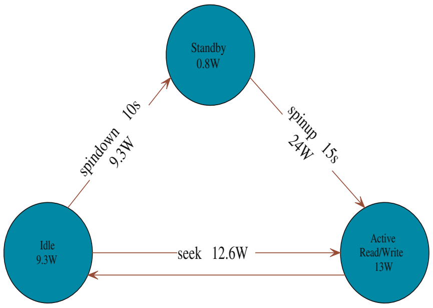


**Fig. 8.** Power consumption of disks in different modes

### 5.1 Hard Disk Characteristics

Table 2 shows the characteristics of hard disk used in the simulation. With the specifications taken from [3] and [25] we built our own hard disk simulation modules. A hard disk is spun down and set into standby mode (see Figure 8) after it has been idle for a fixed period which is called *idleness threshold* [26, 11]. We do not use the recently

**Table 2.** Characteristics of the Hard Disk

| *Description* | *Value* | *Description* | *Value* |
|---|---|---|---|
| Disk model | Seagate ST3500630AS | Standby power | 0.8 Watts |
| Standard interface | SATA | Active power | 13 Watts |
| Rotational speed | 7200 rpm | Seek power | 12.6 Watts |
| Avg. seek time | 8.5 msecs | Spin up power | 24 Watts |
| Avg. rotation time | 4.16 msecs | Spin down power | 9.3 Watts |
| Disk size | 500GB | Spin up time | 15 secs |
| Disk load (Transfer rate) | 72 MBytes/sec | Spin down time | 10 secs |
| Idle power | 9.3 Watts | | |

revised DiskSim simulator [27], that is commonly used in the literature for our simulations because, it still provides only old and small disk models, e.g., 1998's 8GBytes disks, and the number of events needed to handle a file request is highly correlated with file sizes making DiskSim too slow for a realistic data center simulation that involves disks, each of the order of 500 GBytes and tens of thousands of files requiring terabytes/petabytes of total data storage.

## 5.2 Workload Generator

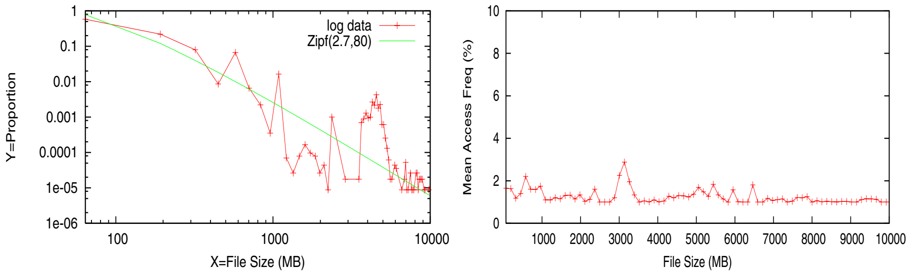
The workload generator supports two different ways to produce file requests. First, the generator can produce requests based on a log of file accesses to a storage system. We extract the distribution of file file sizes and the arrival time of each request from the real workload. Second, the generator can follow a Poisson process to produce requests at a rate $R$ to get files specified in a given list. The sizes of the files in the list are generated based on a Zipf distribution whose probability distribution is given by

$$P(x) = \frac{x^{-K}}{\zeta(N,K)}; \quad \text{where} \quad \zeta(N,K) = \sum_{i=1}^{N} i^{-K}. \tag{1}$$

Also, the generator can control the frequency of requests to each file. To determine reasonable parameters for the Zipf distribution that are close to the actual data accesses, we logged the file requests to the NERSC's High Performance Storage System (HPSS) for 30 days (between May 31 and June 29, 2008). There were 88,631 files accessed in the 115,832 read requests. The mean size of the files requested was 544 MB. This requires 7.56 sec to service a file if these files were to be accessed from a disk storage systems with disk transmission rate of 72MBps. The minimum space required for storing all the requested files is 95 disks. Next we classified the 88,631 files into 80 bins based on their sizes, where the width of each bin is 128MB. We then compute the proportion of the number of files in each bin compared with the total number of files. Figure 9(a) plots these proportions for the 80 bins. Each point $z\langle X,Y\rangle$ in Figure 9(a), represents the proportion Y of files with sizes in the interval $(X-64, X+64]$ in MBytes. As we can see this distribution is closely related to the Zipf distribution because the proportion decreases almost linearly in the log-log scale of the axes. Figure 9(b) shows the relationship between the sizes of files and their corresponding access frequencies. In this analysis of accesses to NERSC datasets, the access frequencies of files are independent of the sizes. We can therefore assume that each file has the same access frequency $f$.

### 5.3 File Dispatcher and Mapping Table

Once a request is generated, the file dispatcher forwards it to the corresponding disk based on the file-to-disk mapping table. Files are randomly mapped to a specific number of disks. The number of disks and idleness thresholds are determined by the procedure proposed in Section 4. For the purpose of comparing the power consumptions of the DPM strategy, we also generated a mapping table that maps files randomly to all disks and fix the idleness threshold at 0, 53, 160 secs or ∞, i.e., without enabling the power saving features of the disk. The time to map a file to disk by the dispatcher is ignored since it is negligible.



(a) Log-log plot of workload distribution (b) Plot of access frequency of workload

**Fig. 9.** Analysis of workload from NERSC

## 6 Experimental Results

### 6.1 Evaluation of the Model

We first evaluate the correctness of our analytical model proposed in Section 3. Only one disk is used in this scenario. The service time of requests has the same distribution and parameters (see Table 2), as those assumed in the analysis. The simulation ends when it has served 30000 requests. Figures 10 and 11 show the normalized power cost of the disk and the response time of requests under different values of arrival rate and idleness threshold respectively. By respectively comparing the two figures with Figures 2 and 3, we can validate our analysis for power cost and response time.

### 6.2 Constraints on Response Time

Next, we examine whether the procedure *SmartIdle* determines the suitable number of active disks and idleness threshold to meet the response time constraints while still saving power compared with our analytical model. Suppose we consider using 100 disks and set $N = 40$ and try to satisfy a response time constraint of 20 seconds. For

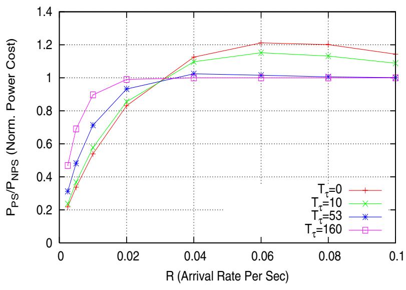


**Fig. 10.** Power cost at different arrival rates

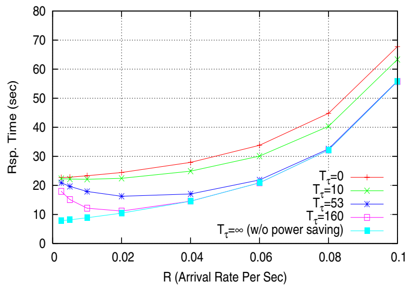


**Fig. 11.** Response times at different arrival rates

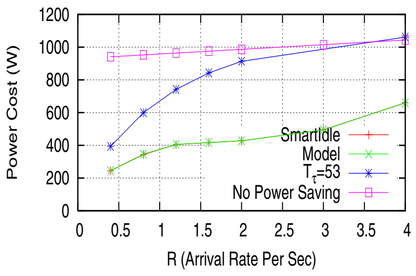


**Fig. 12.** Power savings with constraint $\theta' = 20$ and $N = 40$

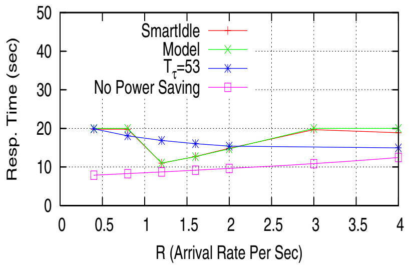


**Fig. 13.** Response time with constraint $\theta' = 20$ and $N = 40$

comparison, we also plotted the response times and power saving ratio when the disk idleness threshold is fixed at 53 seconds, which as mentioned before is the *competitive idleness threshold* in our case.

Figures 12 and 13 show the power saving and response time plots for a response time constraint of 20-sec. The SmartIdle procedure design satisfies the response time constraint while saving 60% of power on average. The *SmartIdle* procedure results in a much shorter response time than the specified constraint for arrival rates ranging from 0.8 to 2.5 and yet saves more power than with fixed 53-sec idleness threshold. In this interval of arrival rates the point $\langle\lambda, \theta'\rangle$ always falls in Area 1 and the SmartIdle procedure suggests that the disks be kept spinning at all times, instead of spinning down after an idleness threshold. Spinning down a disk after a fixed idleness threshold is inefficient since it not only results in a longer response time but also incurs more power cost than simply spinning disk.

## 6.3 Using Trace Logs of Scientific Data Accesses

In this subsection we test whether the SmartIdle procedure can be used to derive the configuration of disks that satisfy the constraint of response time when apply the request arrival rate extracted from a real workload. Suppose we consider the use of 200 disks with the minimum number required set as $N = 96$. The arrival rate of requests is 0.044683. Figure 14 shows the ratio of power savings obtained from using the analytical model, the *SmartIdle* procedure and fixed idleness threshold of 53 secs. The initial

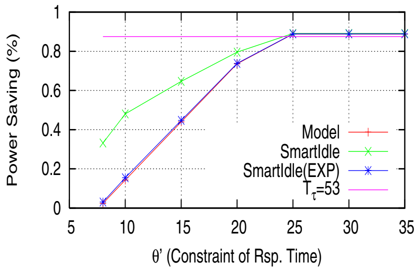


**Fig. 14.** Power savings between SmartIdle and fixed idleness threshold values for NERSC data

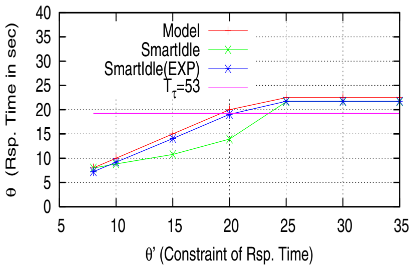


**Fig. 15.** Response times of SmartIdle and fixed idleness threshold values for NERSC data

idleness threshold used varies from 8 to 35 seconds. Figure 15 shows the corresponding response times of obtained in each design configuration. From Figure 15, we find that disks configuration obtained from applying the *SmartIdle* not only meets the constraints, but also provides response time far less than the initial response time constraints ranging from 8 to 25 sec. Further we see that from Figure 14 that we achieve more power savings than that the expected savings achievable from the analytical model.

In addition to the NERSC workload, we also tested our *SmartIdle* procedure with workloads from the BaBar project [28]. The BaBar project is a high energy physics experiment with over 600 world-wide collaborators from 75 institutions in 10 countries. The data for this experiment is stored at the Stanford Linear Accelerator Collider (SLAC) site. There are about 86,378 distinct files stored which will require at least 123 disks of 500GB to store them. The trace log of file requests for Oct 1, 2004 was used in this study. It contained 93,172 read requests and involved 10,735 distinct files. The average arrival rate (per second) of the requests is 1.07838, which is much higher than that in the workload of NERSC. The mean size of files accessed by these requests is 1,235 MB, which requires about 16.5138 sec of mean service time, $E[S]$, and 332.438 sec for E[S$^2$] when the disk transmission rate is 72MBps and a single 32GB LRU cache is deployed in front of all disks.

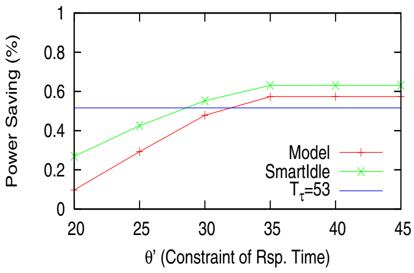


**Fig. 16.** Power savings between SmartIdle and fixed idleness threshold values for BaBar

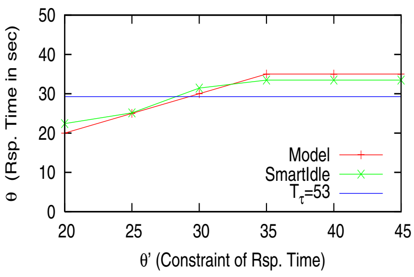


**Fig. 17.** Response times of SmartIdle and fixed idleness threshold values for BaBar data

Compared to the uniformly distributed accesses observed in the NERSC workload (Figure 9(b)), the workload shows 48% of requests accessing files with size larger than 1.6GB. Further observation shows these requests target only 783 files that constitute 7.3% of the files involved in all requests.

Figure 16 shows the ratio of power savings incurred while Figure 17 shows the response time of disks when their idleness threshold are configured by SmartIdle for constraints varying from 20 to 45 seconds. From the two figures, we again find that disk configuration derived from using *SmartIdle* not only meets the constraints of the response times, but further gives a greater saving than expected by the analytical model.

## 7 Conclusion and Future Work

In this paper we developed an analytic model to analyze the interaction of file access workload with a disk system that uses power saving mechanisms. The model allowed us to devise a procedure that allows designers to accurately evaluate the trade-offs between energy consumption and response time. The procedure can be used to determine whether the required response times are achievable by the current system and what are the associated energy costs. The procedure also allows designers to tune the performance of the system by adding or subtracting disks as well as determining idleness thresholds. Using the procedure on simulated data as well as real life work logs showed significant improvement in energy costs over commonly used DPM strategies.

Additional work also needs to be done to make dynamic decisions about migrating files between disks if it is discovered that the arrival rates to disks deviate significantly from the initial estimates used as an input to the SmartIdle procedure. We also plan to investigate our techniques with more real life workloads that include various mixes of read and write requests. In addition, we will also investigate the effects of various caching strategies as we believe that cache size and cache replacement policies may significantly affect the trade-off between power consumption and response time.

## References

1. Barroso, L.A.: The price of performance. Queue 3(7), 48–53 (2005)
2. Gurumurthi, S., Sivasubramaniam, A., Kandemir, M., Franke, H.: Reducing disk power consumption in servers with drpm. Computer 36(12), 59–66 (2003)
3. Seagate: Seagate Barracuda 7200.10 Serial ATA Product Manual. Seagate Technology, Scotts Valley, CA (December 2007)
4. Colarelli, D., Grunwald, D.: Massive arrays of idle disks for storage archives. In: Supercomputing 2002: Proc. ACM/IEEE Conference on Supercomputing, pp. 1–11. IEEE Computer Society Press, Los Alamitos (2002)
5. Weddle, C., Oldham, M., Qian, J., Wang, A.: PARAID: A gear shifting power-aware RAID. ACM Trans. on Storage (TOS) 3(3), 26–28 (2007)
6. Storer, M.W., Greenan, K.M., Miller, E.L.: Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage. In: Proc. 6th USENIX Conf. on File and Storage Technologies (FAST 2008), San Jose, California, pp. 1–16 (Feburary 2008)
7. Xie, T.: Sea: A striping-based energy-aware strategy for data placement in RAID-structured storage system. IEEE Transactions on Computers 57(6), 748–769 (2008)

8. Irani, S., Singh, G., Shukla, S.K., Gupta, R.K.: An overview of the competitive and adversarial approaches to designing dynamic power management strategies. IEEE Trans. VLSI Syst. 13(12), 1349–1361 (2005)
9. Narayanan, D., Donnelly, A., Rowstron, A.: Write off-loading: Practical power management for enterprise storage. In: Proc. 6th USENIX Conf. on File and Storage Technologies (FAST 2008), San Jose, California, pp. 253–267 (Feburary 2008)
10. Zhu, Q., Chen, Z., Tan, L., Zhou, Y., Keeton, K., Wilkes, J.: Hibernator: Helping disk arrays sleep through the winter. In: SOSP 2005: Proc. 20th ACM Symp. on Operating Syst. Principles, pp. 177–190. ACM Press, New York (2005)
11. Pinheiro, E., Bianchini, R.: Energy conservation techniques for disk array-based servers. In: Proc. Int'l. Conf. on Supercomputing (ICS 2004), Saint-Malo, France, June 26 (2004)
12. Guha, A.: Data archiving using enhanced maid (massive array of idle disks), May 15-18 (2006)
13. Guha, A.: A new approach to disk-based mass storage systems. In: 12th NASA Goddard - 21st IEEE Conf. on Mass Storage Syst. and Tech., College Park, Maryland (2004)
14. Karlin, A.R., Manasse, M.S., McGeoch, L.A., Owicki, S.: Competitive randomized algorithms for non-uniform problems. In: SODA 1990: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms, pp. 301–309. Society for Industrial and Applied Mathematics, Philadelphia (1990)
15. Irani, S., Gupta, R., Shukla, S.: Competitive analysis of dynamic power management strategies for systems with multiple power savings states. In: DATE 2002: Proceedings of the conference on Design, automation and test in Europe, p. 117. IEEE Computer Society, Washington (2002)
16. Ramanathan, D., Irani, S., Gupta, R.: Latency effects of system level power management algorithms. In: ICCAD 2000: Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design, pp. 350–356. IEEE Press, Piscataway (2000)
17. Rivoire, S., Shah, M.A., Ranganathan, P., Kozyrakis, C.: Joulesort: a balanced energy-efficiency benchmark. In: SIGMOD 2007: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pp. 365–376. ACM, New York (2007)
18. Poess, M., Nambiar, R.O.: Energy cost, the key challenge of today's data centers: a power consumption analysis of tpc-c results. Proc. VLDB Endow. 1(2), 1229–1240 (2008)
19. Lu, L., Varman, P.: Workload decomposition for power efficient storage systems. In: Proc. Workshop on Power Aware Computing and Systems (HotPower 2008), San Diego, CA, USA (December 2008)
20. Harizopoulos, S., Shah, J.M.M.A., Ranganathan, P.: The new holy grail of data management systems research. In: CIDR 2009: Proc. of the 4th Biennial Conf. on Innovative Data Syst. Research. ACM, New York (2009)
21. Lang, W., Patel, J.M.: Towards ecofriendly database management systems. In: CIDR 2009: Proc. of the 4th Biennial Conf. on Innovative Data Syst. Research. ACM, New York (2009)
22. Bisson, T., Brandt, S.A., Long, D.D.E.: A hybrid disk-aware spin-down algorithm with I/O subsystem support. In: Proc. Perf., Comput., and Com. Conf. (IPCCC 2007), New Orleans, LA, pp. 236–245 (May 2007)
23. Lee, L.W., Scheuermann, P., Vingralek, R.: File assignment in parallel I/O systems with minimal variance of service time. IEEE Transactions on Computers 49(2), 127–140 (2000)
24. SourceForge .net: SimPy Simulation Package in Python, `http://simpy.sourceforge.net/archive.htm`
25. Zedlewski, J., Sobti, S., Garg, N., Zheng, F., Krishnamurthy, A., Wang, R.: Modeling hard-disk power consumption. In: FAST 2003: Proc. 2nd USENIX Conf. on File and Storage Tech., Berkeley, CA, USA, pp. 217–230. USENIX Association (2003)
26. Cunha, C., Bestavros, A., Crovella, M.: Characteristics of www client-based traces. Technical report, Boston University, Boston, MA, USA (1995)

27. Bucy, J., Schindler, J., Schlosser, S., Ganger, G.: The disksim simulation environment
28. BaBar: The babar collaboration, `http://www.slac.stanford.edu/bfroot/`

## Appendix A: Derivation of Results for Analytical Model

Expressions for computing the power costs and the mean response times are given below.

***Case 1,*** $t < T_\tau$ **:**
$P^1 = \text{Prob}\{t < T_\tau\}$; $E[G^1_{PS}] = P_\tau E[T_1] + P_a E[T_a]$; and $E[T^1_{PS}] = E[T_1] + E[T_a]$;
where E[$T_1$] is $E[t]$ in Case 1 and can be written as $E[T_1] = \int_0^{T_\tau} tp(t)dt / \int_0^{T_\tau} p(t)dt$ where $p(t)$ is the probability density function (pdf) of $t$.

***Case 2,*** $T_\tau < t < T_\tau + T_d$**:**
$P^2 = \text{Prob}\{T_\tau < t < T_\tau + T_d\}$; $E[E^2_{PS}] = P_\tau T_\tau + G_{du} + P_a E[T_2]$; and $E[T^2_{PS}] = T_\tau + T_d + T_u + E[T_2]$;
where $E[T_2]$ is the busy period in Case 2 and can be written as

$$E[T_2] = \frac{\int_{T_\tau}^{T_\tau+T_d} E[f_{T_a}(T_d + T_u - (t - T_\tau))]p(t)dt}{\int_{T_\tau}^{T_\tau+T_d} p(t)dt}.$$

***Case 3,*** $(T_\tau + T_d) < t$**:**
$P^3 = \text{Prob}\{T_\tau + T_d < t\}$; $E[E^3_{PS}] = P_\tau T_\tau + P_d T_d + P_{sby} E[T_3] + P_u T_u + P_a E[f_{T_a}(T_u)]$; and $E[T^3_{PS}] = T_\tau + T_d + E[T_3] + T_u + E[f_{T_a}(T_u)]$;
where $T_3$ is standby time, $E[\text{T}_{sby}]$, in Case 3 and can be written

$$E[T_3] = \frac{\int_{T_\tau+T_d}^{\infty} (t - T_\tau - T_d) p(t)dt}{\int_{T_\tau+T_d}^{\infty} p(t)dt}.$$

### Mean Response Time

Similarly, by the above model for $D_{PS}$, we can calculate the mean sojourn time $\theta$ of a request, i.e. its response time, by averaging the $E[\theta]$ of each case as

$$E[\theta_{PS}] = \sum_{i=1}^{3} E[\theta^i] P^i;$$

where $P^i$ and $E[\theta^i]$ are the probability and the mean request sojourn time, respectively during the cycle of case $i$. Besides, since a $D_{NPS}$ does not spin down, we can simply regard its $T_\tau$ as $\infty$ and then get $E[\theta_{NPS}] = E[\theta^1]$. Next, recall that the sojourn time in M/G/1 [2], is

$$E[\theta] = \frac{\rho}{1-\rho} \frac{E[S^2]}{2E[S]} + E[S]; \tag{2}$$

and in M/G/1 with setup time $X$, the mean sojourn time $E[\theta_x]$ is

$$E[\theta_X] = \frac{\rho}{1-\rho} \frac{E[S^2]}{2E[S]} + \frac{\lambda^{-1}}{\lambda^{-1} + E[X]} + \frac{E[X]}{\lambda^{-1} + E[X]} \frac{E[X^2]}{2E[X]} + E[S]. \tag{3}$$

Then, according to the above two equations, we get $E[\theta^1]$, $E[\theta^2]$ and $E[\theta^3]$ as follows:

***Case 1,*** $t < T_\tau$**:**
Based on equation 2, we have $E[\theta^1] = (\rho/(1-\rho))(E[S^2]/2E[S]) + E[S]$

***Case 2,*** $T_\tau < t < T_\tau + T_d$**:**

From Case 2 of Subsection 3.2, we have that the setup time $X_2$, of Case 2 is $X_2 = T_2 = T_\tau + T_d + T_u - t$. So we get

$$E[X_2] = \frac{\int_{T_\tau}^{T_\tau+T_d} (T_i + T_d + T_u - t) p(t) dt}{\int_{T_\tau}^{T_i+T_d} p(t) dt}. \tag{4}$$

$$E[X_2^2] = \frac{\int_{T_\tau}^{T_\tau+T_d} (T_\tau + T_d + T_u - t)^2 p(t) dt}{\int_{T_\tau}^{T_\tau+T_d} p(t) dt}. \tag{5}$$

Then, we can express $E[\theta_2]$ from 3 by substituting terms with 4 and 5.

***Case 3,*** $T_\tau + T_d < t$**:**

Because the setup time in Case 3 is $X_3 = T_u$, we have

$$E[\theta^3] = \frac{\rho}{(1-\rho)} \frac{E[S^2]}{2E[S]} + \frac{\lambda^{-1}}{\lambda^{-1} + T_u} + \frac{T_u}{\lambda^{-1} + T_u} \frac{T_u}{2} + E[S].$$

# Data Parallel Bin-Based Indexing for Answering Queries on Multi-core Architectures

Luke J. Gosink[1], Kesheng Wu[2], E. Wes Bethel[3], John D. Owens[1], and Kenneth I. Joy[1]

[1] Institute for Data Analysis and Visualization (IDAV)
One Shields Avenue, University of California, Davis, CA 95616-8562, U.S.A.
{ljgosink,joy}@ucdavis.edu, jowens@ece.ucdavis.edu
[2] Scientific Data Management Group, Lawrence Berkeley National Laboratory,
1 Cyclotron Road, Berkeley, CA 94720, U.S.A.
kwu@lbl.gov
[3] Visualization Group, Lawrence Berkeley National Laboratory,
1 Cyclotron Road, Berkeley, CA 94720, U.S.A.
ewbethel@lbl.gov

**Abstract.** The multi-core trend in CPUs and general purpose graphics processing units (GPUs) offers new opportunities for the database community. The increase of cores at exponential rates is likely to affect virtually every server and client in the coming decade, and presents database management systems with a huge, compelling disruption that will radically change how processing is done. This paper presents a new parallel indexing data structure for answering queries that takes full advantage of the increasing thread-level parallelism emerging in multi-core architectures. In our approach, our Data Parallel Bin-based Index Strategy (DP-BIS) first bins the base data, and then partitions and stores the values in each bin as a separate, bin-based data cluster. In answering a query, the procedures for examining the bin numbers and the bin-based data clusters offer the maximum possible level of concurrency; each record is evaluated by a single thread and all threads are processed simultaneously in parallel.

We implement and demonstrate the effectiveness of DP-BIS on two multi-core architectures: a multi-core CPU and a GPU. The concurrency afforded by DP-BIS allows us to fully utilize the thread-level parallelism provided by each architecture–for example, our GPU-based DP-BIS implementation simultaneously evaluates over 12,000 records with an equivalent number of concurrently executing threads. In comparing DP-BIS's performance across these architectures, we show that the GPU-based DP-BIS implementation requires significantly less computation time to answer a query than the CPU-based implementation. We also demonstrate in our analysis that DP-BIS provides better overall performance than the commonly utilized CPU and GPU-based projection index. Finally, due to data encoding, we show that DP-BIS accesses significantly smaller amounts of data than index strategies that operate solely on a column's base data; this smaller data footprint is critical for parallel processors that possess limited memory resources (e.g. GPUs).

## 1 Introduction

Growth in dataset size significantly outpaces the growth of CPU speed and disk throughput. As a result, the efficiency of existing query processing techniques is greatly challenged [1,2,3]. The need for accelerated I/O and processing performance forces many researchers to seek alternative techniques for query evaluation. One general trend is to develop highly parallel methods for the emerging parallel processors, such as multi-core processors, cell processor, and the general-purpose graphics processing units (GPU) [4]. In this paper, we propose a new parallel indexing data structure that utilizes a Data Parallel Bin-based Index Strategy (DP-BIS). We show that the available concurrency in DP-BIS can be fully exploited on commodity multi-core CPU and GPU architectures.

The majority of existing parallel database systems work focuses on making use of multiple loosely coupled clusters, typified as shared-nothing systems [5,6,7,8,9,10]. Recently, a new parallel computing trend has emerged. These type of parallel machines consist of multiple tightly-coupled processing units, such as multi-core CPUs, cell processors, and general purpose GPUs. The evolution of such machines in the coming decade is to support a tremendous number of concurrent threads working from a shared memory. For example, NVIDIA's 8800 GTX GPU–the GPU used in this work–has 16 multiprocessors, each of which supports 768 concurrent execution threads. Combined, these multiprocessors allow the GPU to manage over 12,000 concurrent execution threads. Fully utilizing such thread-level parallelism on a shared memory system requires a different set of query processing algorithms than on shared-nothing systems.

A number of researchers have successfully demonstrated the employment of GPUs for database operations [11,12,13,14]. Among the database operations, one of the basic tasks is to select a number of records based on a set of user specified conditions, e.g., "SELECT: records FROM: combustion_simulation WHERE: pressure $>$ 100." Many GPU-based works that process such queries do so with a projection of the base data [15,11]. Following the terminology in literature, we use the term *projection index* to describe this method of sequentially and exhaustively scanning all base data records contained in a column to answer a query [16]. On CPUs, there are a number of indexing methods that can answer queries faster than the projection index [17,18,19], but most of these indexing methods do not offer high enough levels of concurrency to take full advantage of a GPU. DP-BIS fully utilizes the GPU's parallelism when answering a selection query; each thread on the GPU is used to independently access and evaluate an individual record. This one-to-one mapping of threads-to-records lets DP-BIS process large amounts of data with 12,000 concurrent parallel operations at any one time.

Though GPUs offer tremendous thread-level parallelism, their utility for database tasks is limited by a small store of resident memory. For example, the largest amount of memory available on NVIDIA's Quadro FX GPU is currently 4.0 GB, which is much too small to hold projections of all columns from a dataset of interest [1,2,3]. DP-BIS presents one method for ameliorating the challenges imposed by limited GPU memory. The DP-BIS index uses a form of data encoding that is implemented through a multi-resolution representation of the base data information. This encoding effectively reduces the amount of data we must access and transfer when answering a query. As a result of the encoding, we can query dataset sizes that would otherwise not fit into the memory footprint of a GPU. Additionally, by transferring smaller amounts of data when answering a query, we utilize data bus bandwidth more efficiently.

In the DP-BIS approach, we bin the base data for each column. We augment each column's binned index by generating a corresponding Data Parallel Order-preserving Bin-based Cluster (OrBiC). To resolve a query condition on a column, we first determine the boundaries of the query. Consider an example. For range conditions such as "*pressure* $> 100$", we determine the bin whose range captures the constraint "100". In this example, assume that the value "100" is contained in the value range captured by $\text{bin}_{17}$. We refer to bins that capture one of the query's constraints as "boundary bins". In our example, records contained in bins less than the boundary bin (i.e. $\text{bin}_0 \rightarrow \text{bin}_{16}$) fail the query. Correspondingly, records contained in bins greater than the boundary bin pass the query. Boundary bin records can't be characterized by their bin number alone, they must be evaluated by their base data value. We call the records in the boundary bin the candidates and the process of examining the candidate values the candidate check [20]. Our strategy for answering a selection query is very similar to that of a bitmap indexing strategy [21, 22, 23]. A central difference is that bitmap-based strategies indicate the record contents of each bin with a single bitmap vector. These bitmap vectors can then be logically combined to help form the solution to a query. In contrast, we directly access the bin number for any given record from an encoded data table.

The Data Parallel OrBiC structure we use during our candidate check procedure provides an efficient way to extract and send boundary bin data from the CPU to the GPU. Additionally, this structure facilitates a rapid, concurrent way for GPU threads to access this data. Altogether, to answer a query, we access the bin numbers and the base data values of the records in boundary bins. The total data contained in both these data structures is much smaller than the column projections used by other strategies that employ the GPU to answer a query. Additionally, the procedure for examining the bin numbers and the process of performing the candidate checks offer the same high level of concurrency as the GPU projection index.

In our work we assume that the base data will not (or seldom) be subjected to modification. This assumption too is made by other research database management systems that operate on large data warehouses that contain read-only data: e.g. MonetDB [24], and C-Store [25]. In addition to such database management systems, many scientific applications also accumulate large amounts of data that is never modified or subjected to transactions [26].

Finally, we specifically utilize and emphasize the GPU in our work because it provides some of the highest amounts of thread-level parallelism available in existing multi-core architectures. To this extent we view the GPU as a representative case of where multi-core architectures are evolving with respect to thread-level parallelism and processing performance. In summary, this paper makes the following three contributions.

– We introduce a data parallel bin-based indexing strategy (DP-BIS) for answering selection queries on multi-core architectures. The concurrency provided by DP-BIS fully utilizes the thread-level parallelism emerging in these architectures in order to benefit from their increasing computational capabilities.
– We present the first strategy for answering selection queries on a GPU that utilizes encoded data. Our encoding strategy facilitates significantly better utilization of data bus bandwidth and memory resources than GPU-based strategies that rely exclusively on base data.

– We implement and demonstrate DP-BIS's performance on two commodity multi-core architectures: a multi-core CPU and a GPU. We show in performance tests that both implementations of DP-BIS outperform the GPU and CPU-based projection index with respect to total query response times. We additionally show that the GPU-based implementation of DP-BIS outperforms all index strategies with respect to computation-based times.

## 2 Background and Related Work

### 2.1 Related Bitmap Index Work

The data stored in large data warehouses and the data generated from scientific applications typically consists of tens to hundreds of attributes. When answering queries that evaluate such high-dimensional data, the performance of many indexing strategies diminishes due to *the curse of dimensionality* [27]. The bitmap index is immune to this curse and is therefore known to be the most efficient strategy for answering ad hoc queries over such data [28]. For this reason, major commercial database systems utilize various bitmap indexing strategies (e.g. ORACLE, IBM DB2, and Sybase IQ).

Another trait of the bitmap index is that storage concerns for indices are ameliorated through specialized compression strategies that both reduce the size of the data and that facilitate the efficient execution of bitwise Boolean operations [29]. Antoshenkov et al. [21,22] present a compression strategy for bitmaps called the Byte-aligned Bitmap Code (BBC) and show that it possess excellent overall performance characteristics with respect to compression and query performance. Wu et al. [23] introduce a new compression method for bitmaps called Word-Aligned Hybrid (WAH) and show that the time to answer a range query using this bitmap compression strategy is optimal; the worse case response time is proportional to the number of hits returned by the query. Recent work by Wu et al. [30] extends the utility of the bitmap index. This work introduces a new Order-preserving Bin-based Clustering structure (OrBiC), along with a new hybrid-binning strategy for single valued bins that helps the bitmap index overcome *the curse of cardinality*; a trait where both index sizes and query response time increase in the bitmap index as the number of distinct values in an attribute increases.

Sinha and Winslet [31] successfully demonstrate parallelizable strategies for binning and encoding bitmap indexes, compressing bitmap vectors, and answering selection queries with compressed bitmap vectors. The content of their work focuses on supporting bitmap use in a highly parallel environment of multiple loosely-coupled, shared-nothing systems. In contrast, our work addresses the challenges of supporting bin-based indexing on the newly emerging, tightly-coupled architectures that possess tremendous thread-level parallelism; for example the graphics processor unit (GPU).

The basic attributes of the binned bitmap index (bin-based indexing, the use of simple boolean operators to answer selection queries, etc.) can be implemented in a highly parallel environment. For this reason, our new Data Parallel Bin-based Indexing Strategy (DP-BIS) follows the general structure of a binned bitmap index. Unfortunately, bitmap compression strategies, even the parallelizable strategies of Sinha and Winslet [31], do not support enough concurrency to take advantage of the thread-level parallelism offered by tightly-coupled architectures like GPUs. Thus one of the first objectives in our work is to develop a compression strategy, based upon the binning techniques of the binned bitmap index, that supports high levels of concurrency and reduces the amount of data required to answer a query.

## 2.2 Related GPU-Database Work

GPUs have been used to help support and accelerate a number of database functions [11, 32, 33, 12, 13, 14], as well as numerous general purpose tasks [34, 35]. Sun et al. [15] present a method for utilizing graphics hardware to facilitate spatial selections and intersections. In their work, they utilize the GPU's hardware-accelerated color blending facilities to test for the intersection between two polygons in screen space.

Working within the constraints of the graphics API for fragment shaders, Govindaraju et al. [11] present a collection of powerful algorithms on commodity graphics processors for performing the fast computation of several common database operations: conjunctive selections, aggregations, and semi-linear queries. This work also demonstrates the use of the projection index to answer a selection query. Additionally, Govindaraju et al. [12] present a novel GPU-based sorting algorithm to sort billion-record wide databases. They demonstrate that their "GPUTeraSort" outperforms the Indy PennySort1 record, achieving the best reported price-for-performance on large databases.

More recent GPU-database work utilizes powerful, new general purpose GPU hardware that is supported by new data parallel programming languages (see Section 3). These hardware and software advances allow for more complex database primitives to be implemented on the GPU. Fang et al. [13] implement the CSS-Tree in the software GPUQP. This work characterizes how to utilize the GPU for query co-processing, unfortunately there is no performance data published about the implementation.

Lieberman et al. [36] implement an efficient similarity join operation in CUDA. Their experimental results demonstrate that their implementation is suitable for similarity joins in high-dimensional datasets. Additionally, their method performs well when compared against two existing similarity join methods.

He et al. [34] improve the data access locality of multi-pass, GPU-based gather and scatter operations. They develop a performance model to optimize and evaluate these two operations in the context of sorting, hashing, and sparse matrix-vector multiplication tasks. Their optimizations yield a 2-4X improvement on GPU bandwidth utilization and 30–50% improvement on performance times. Additionally, their optimized GPU-based implementations are 2-7X faster than optimized CPU counterparts. He et al. [14] present a novel design and implementation of relational join algorithms: non-indexed and indexed nested loops, sort-merge, and hash joins. This work utilizes their bandwidth optimizations [34], and extends the work of Fang et al. [13]. They support their algorithms with new data-parallel primitives for performing map, prefix-scan and split tasks. Their work achieves marked performance improvements over CPU-based counterparts; GPU-based join algorithms are 2-7X faster than CPU-based approaches.

GPU-based strategies that address how to answer a selection query have yet to address the significant limitations imposed by the GPU's small memory, and those imposed by the data buses that transfer data to the GPU. To the best of our knowledge, all relevant literature utilizes algorithms that operate on a column's base data (i.e. non-compressed data). Utilizing base data severely restricts the amount of data the GPU can process. Further, streaming large amounts of base data to the GPU can impede the processing performance of many GPU-based applications. More specifically, GPU processing performance can rapidly become bottlenecked by data transfer rates if these transfer rates are not fast enough to keep the GPU supplied with new data. This bottleneck event occurs on GPUs whenever the arithmetic intensity of a task is low; the process of answering a simple range query falls into this classification.

In the following sections, we introduce some basic GPU fundamentals, as well as the languages that support general purpose GPU programming. We then introduce our Data Parallel Bin-based Indexing Strategy (DP-BIS) and show how it directly addresses the challenges of limited GPU-memory and performance-limiting bus speeds with a fast, bin-based encoding technique. This work is the first GPU-based work to present such an approach for answering queries. We also show how our binning strategy enables DP-BIS to support a high level of concurrency. This concurrency facilitates a full utilization of the parallel processing capabilities emerging in multi-core architectures.

## 3 GPUs and Data Parallel Programming Languages

Recent GPU-database works utilize powerful new data parallel programming languages like NVIDIA's CUDA [37], and OpenCL. These new programming languages eliminate the long standing tie of general-purpose GPU work with restrictive graphics-based APIs (i.e. fragment/shader programs). Further, the GPUs supporting these languages also facilitate random read and write operations in GPU memory—scatter I/O operations are essential for GPUs to operate as a general-purpose computational machine.

The functional paradigm of these programming languages views the GPU as a co-processor to the CPU. In this model, the programmer writes two separate kernels for a general purpose GPU (GPGPU) application: code for the GPU kernel and the code for the CPU kernel. Here the CPU kernel must proceed through three general stages.

1. Send a request to the GPU to allocate necessary input and output data space in GPU memory. The CPU then sends the input data (loaded from CPU memory or hard disk) to the GPU.
2. Call the GPU kernel. When the CPU kernel calls a GPU kernel, the CPU's kernel suspends and control transfers to the GPU. After processing its kernel, the GPU kernel terminates and control is transferred back to the CPU.
3. Retrieve the output data from the GPU's memory.

From a high level, the GPU kernel serves as a sequence of instructions that describes the logic that will direct each GPU thread to perform a specific set of operations on a unique data element. The kernel thus enables the GPU direct the concurrent and simultaneous execution of all GPU threads in a SIMT (single-instruction, multiple-thread) workflow. The GPU executes its kernel (step two above) by first creating hundreds to thousands of threads—the number of threads is user specified and application dependent. During execution, small groups of threads are bundled together and dynamically dispatched to one of the GPU's numerous SIMD multiprocessors. These thread bundles are then delegated by the multiprocessor to one of its individual processors for evaluation. At any given clock cycle, each processor will execute the same kernel-specified instruction on a thread bundle, but each thread will operate on different data.

With respect to memory resources, each GPU multiprocessor contains a set of dedicated registers, a store of read-only constant and texture cache, and a small amount of shared memory. These memory types are shared between the individual processors of a multiprocessor. In addition to these memory types, threads evaluated by a processor may also access the GPU's larger, and comparatively slower, global memory.

There are two important distinctions to make between GPU threads and CPU threads. First, there is no cost to create and destroy threads on the GPU. Additionally, GPU multiprocessors perform context switches between thread bundles (analogous to process switching between processes on a CPU) with zero latency. Both of these factors enable the GPU to provide its thread-level parallelism with very low overhead.

## 4 A Data Parallel Bin-Based Indexing Strategy (DP-BIS)

### 4.1 Overview

To effectively utilize a GPU, an indexing data structure must provide high levels of concurrency to fully benefit from the GPU's large number of concurrent execution threads, and make effective use of the GPU's relatively small memory. In this section we explain our new DP-BIS method and show how it successfully addresses these requirements by integrating two key strategies: data binning (Section 4.2) and the use of Data Parallel Order-preserving Bin-based Clusters (OrBiC) (Section 4.3).

When answering a query, the binning strategy we utilize significantly reduces the amount of data we must access, transfer, and store on the GPU. The Data Parallel OrBiC structure we employ ensures that candidate checks only access the base data of the boundary bins. The concurrency offered by both of these data structures facilitates full utilization of the GPU's thread-level parallelism. In this approach, DP-BIS builds one index for each column in a database, where each index consists of an encoded data table (i.e. the bin numbers), and a Data Parallel OrBiC structure. When answering a simple range query with DP-BIS, we access the encoded data table, and the base data of two bins (the boundary bins) from our data parallel OrBiC structure.

### 4.2 Base Data Encoding

The index construction process begins by binning all of the base data records contained in a single column. To minimize data skew in our binning strategy, we select the bin boundaries so that each bin contains approximately the same number of records. In cases where the frequency of a single value exceeds the allotted record size for a given bin, a *single-valued bin* is used to contain all records corresponding to this one value. This technique to address data skew is consistent with other binning strategies [30]. We then encode the base data by representing each base data record with its associated bin number. Figure 1 (Step 1) in Section 4.3 shows an example of this encoding. In later discussions, we refer to the bin numbers as low-resolution data and the column's base data as full-resolution data. We always utilize 256 bins in our encoding procedure. As we now show, the amount of data generated by using this number of bins facilitates near-optimal usage of bus bandwidth and GPU memory space when answering a query.

**Table 1.** This table presents the total benefit for DP-BIS to utilize a specific number of bins in its encoding strategy. All values in column two, three, and four are given in terms of a percentage of the total full-resolution data (assuming 32-bits are utilized to represent each full-resolution record). Note the *Boundary Bin Size* reflects the cost for *two* boundary bins. From this table we see that the use of 256 bins reduces the amount of data we must transfer and store by over 74%.

| *Number of Bins* | *Low-Resolution Size(%)* | *Boundary Bin Size(%)* | *Total Data Size(%)* |
|---|---|---|---|
| $2^{32} = 4294967296$ | 100.0 | 0.0 | 100.0 |
| $2^{16} = 65536$ | 50.0 | $100.0 \times \frac{2}{65536} = 0.003$ | $50.0 + 0.003 = 50.0$ |
| **$2^8$=256** | **25.0** | **$100.0 \times \frac{2}{256} = 0.78$** | **$25.0 + 0.78 = 25.8$** |

Assume that all full-resolution data is based on 32-bit values, and that there are N records in a given database column. If we use $x$ bits to represent each bin, we can then create $2^x$ bins where each bin will contain, on average, $\frac{N}{2^x}$ records. The total size of the low-resolution data will then be $x \times N$ bits. The candidate data for each boundary bin, assuming each row-id can be stored in 32-bits, will consist of $\frac{32 \times N}{2^x}$ bits for row-identifiers, and $\frac{32 \times N}{2^x}$ bits for data values. The total number of bits (written as $B$ below) we utilize to answer a simple range query is therefore:

$$B = \underbrace{x \times N}_{Low-Resolution\ Bits} + \underbrace{(4 \times \frac{32 \times N}{2^x})}_{Candidate\ Check\ Bits\ for\ Boundary\ Bins} \quad (1)$$

Note that the candidate check bit cost for boundary bin data is based on two boundary bins; this data size represents the more typical, and expensive, workload for answering a simple range query. Taking the derivative of $B$ with respect to $x$, we get:

$$\frac{dB}{dx} = N - (128 \times N \times \frac{\ln 2}{2^x}) \quad (2)$$

By setting this derivative to 0 and solving for $x$, we compute the optimal number of bits to use for our strategy:

$$B_{min} = 7 + \log_2(\ln(2)) \approx 6.4712(bits) \quad (3)$$

In our encoding strategy, bins can be represented with either 32-bits, 16-bits, or 8-bits; these are the most easy and efficient data sizes for GPUs and CPUs to evaluate. Use of alternate data sizes, like the "optimal" 6-bit data type we have derived in Equation 3, are not convenient for GPU-processing. The closest integer type that is conveniently supported on the GPU is the 8-bit integer. Therefore we use 8-bit integers to represent bin numbers and we utilize 256 bins in our encoding strategy.

Table 1 illustrates the benefit of using 256 bins from a less formal standpoint. This table shows realized data transfer costs for answering a simple range query based on data types efficient for CPU and GPU computation. The last row validates Equation 3; 8-bit bins provide the best encoded-based compression by reducing the amount of data that must be accessed, transferred, and stored on the GPU by over 74% .

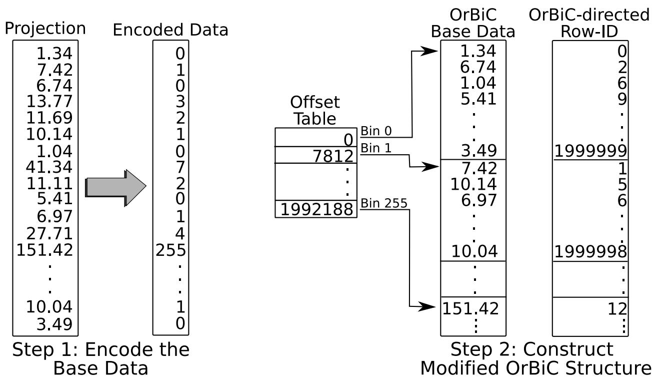


**Fig. 1.** This figure shows the two-step DP-BIS index construction process. The first step encodes a column's full-resolution base data (Section 4.2). The second step (Section 4.3) utilizes this same full-resolution information to generate a modified (i.e. Data Parallel) OrBiC Structure.

## 4.3 Extending OrBiC to Support Data Parallelism

Wu et al. [30] introduce an Order-preserving Bin-based Clustering (OrBiC) structure; this structure facilitates highly efficient candidate checks for bitmap-based query evaluation strategies. Unfortunately, the OrBiC structure does not offer enough concurrency to take advantage of the GPU's parallelism. In this section, we present the constructs of the original OrBiC data structure, and then address how we extend this index to provide greater levels of concurrency.

In the approach presented by Wu et al., the full-resolution data is first sorted according to the low-resolution bin numbers. This reordered full-resolution table is shown as the "OrBiC Base Data" table in Figure 1. In forming this table, each bin's start and end positions are stored in an offset table. This offset table facilitates contiguous access to all full-resolution data corresponding to a given bin.

We extend the work of Wu et al. by building an OrBiC-directed table; this is the "OrBiC-directed Row-ID" table in Figure 1. This table holds row-identifier information for the full-resolution data records. The appended "directed" statement refers to the fact that the ordering of this table is directed by the ordering of the OrBiC Base Data table. With consistent ordering between these tables, start and end locations for a given bin in the offset table provide contiguous access to both the full-resolution data contained in this bin and the correct row-identifier information for each of the bin's records.

The OrBiC-directed row-identifier table facilitates data parallelism by addressing a fundamental difference between our data parallel bin-based strategy and the bitmap work of Wu et al. [30]. Specifically, Wu et al. create a single bitmap vector for each bin in the OrBiC Base Data table. As the bitmap vector associated with a given bin stores the bin's row-identifier information implicitly, their procedure does not need to keep track of the row-identifiers. In our case such a strategy is not inherently parallelizable. We thus employ an explicit representation of the row-identifier information by storing them in the OrBiC-directed Row-ID table. Using this table, threads can simultaneously and in parallel perform candidate checks on all records in a given boundary bin.

**Algorithm 1**
GPU Kernel for Low-Resolution Data

```
Require: Integer lowBinNumber, Integer highBinNumber, Integer [] lowResolution
 1: position ← ThreadID
 2: binNum ← lowReslution[position]
 3: if (binNum > lowBinNumber) then
 4:     if (binNum < highBinNumber) then
 5:         Sol[position] ← TRUE
 6:     end if
 7: end if
 8: if (binNum < lowBinNumber) then
 9:     Sol[position] ← FALSE
10: end if
11: if (binNum > highBinNumber) then
12:     Sol[position] ← FALSE
13: end if
```

**Algorithm 2**
GPU Kernel for Candidate Checks

```
Require: Float lowReal, Float highReal, Float [] fullResolution, Integer [] rowID
 1: position ← ThreadID
 2: recordVal ← fullReslution[position]
 3: record_RowID ← rowID[position]
 4: if (recordVal > lowReal) then
 5:     if (recordVal < highReal) then
 6:         Sol[record_RowID] ← TRUE
 7:     end if
 8: end if
 9: if (recordVal < lowReal) then
10:     Sol[record_RowID] ← FALSE
11: end if
12: if (recordVal > highReal) then
13:     Sol[record_RowID] ← FALSE
14: end if
```

### 4.4 DP-BIS: Answering a Query

In this work, we focus on using DP-BIS to solve simple and compound range queries. Range queries in general are a common database query expressed as a boolean combination of two simple predicates: $(100.0 \leq X)$ AND $(X \leq 250)$, or alternatively $(100.0 \leq X \leq 250)$. Compound range queries logically combine two or more simple range queries using operators such as AND, and OR: $(X \leq 250)$ AND ( Y $\leq 0.113$).

Strategies that answer range queries efficiently and rapidly are a crucial underpinning for many scientific applications. For example, query-driven visualization (QDV) integrates database technologies and visualization strategies to address the continually increasing size and complexity of scientific data [38, 39, 40]. In QDV, large data is intelligently pared down by user-specified selection queries, allowing smaller, more meaningful subsets of data to be efficiently analyzed and visualized.

**Simple Range Queries.** The DP-BIS process for answering a simple range query consists of three stages: load necessary input data onto the GPU, execute the GPU kernel, and download the output data (i.e. the query's solution) from the GPU to the CPU. The input for this process consists of a single low-resolution database column, all necessary full-resolution record and row-identifier data, and two real values that will be used to constrain the column. The process returns a boolean bit-vector—a boolean column with one entry per data record that indicates which records have passed the query.

Given a query, the CPU kernel first accesses the appropriate low-resolution data column from disk. Next, space is allocated in GPU memory to hold both this data as well the query's solution. After allocating memory, and sending the low-resolution data to the GPU, the CPU kernel proceeds by identifying the boundary bins of the query. The query's boundary bins are the bins whose ranges contain the query's real-valued constraints. The CPU kernel uses these bin numbers as an index into the OrBiC offset table. Values in the offset table provide the start and end locations in the OrBiC Base Data, and Row-ID tables for the candidate record's full-resolution data and corresponding row-identifiers. After the candidate data is sent to the GPU, the CPU kernel then calls the necessary GPU kernels.

**Table 2.** This table shows the required changes to make to Algorithms 1 and 2 to form the logic-based kernels DP-BIS uses to answer a compound range queries

| Logic-Based *Kernel* | *Algorithm 1 line 1.4, and Algorithm 2 line 2.5 change to:* | *Algorithm 1 line 1.7, and Algorithm 2 line 2.8 change to:* |
|---|---|---|
| AND | "*Sol*[x] ← *Sol*[x]" | "*Sol*[x] ← **FALSE**" |
| OR | "*Sol*[x] ← **TRUE**" | "*Sol*[x] ← *Sol*[x]" |

The first GPU kernel, shown in Algorithm 1, processes the column's low-resolution data. In setting up this kernel, the CPU instructs the GPU to create one thread for each record in the column. The CPU then calls the GPU kernel, passing it the boundary bin numbers; these boundary bin numbers enable threads to answer an initial low-resolution query. At launch time, each thread first determines its unique thread identifier[1]. Threads use their identifier to index into the *lowResolution* data array (line 2); this array is the low-resolution data column loaded earlier by the CPU. The thread characterizes its record as passing or failing depending on whether the record's bin number lies interior, or exterior to the boundary bins (lines 3, 4, 8, and 9). The answer to each thread's query is written to the query's solution space in GPU memory. This space, previously allocated by the CPU kernel, is shown in Algorithm 1 as *Sol*[].

The next GPU kernel, shown in Algorithm 2, performs a candidate check on all records contained in a given boundary bin. In our approach we launch the candidate check kernel twice: once for the lower boundary and once for the higher boundary bins.

The candidate check kernel is similar to the previous GPU kernel. Thread identifiers enable each thread to index into the *Full-Resolution* and *rowID* arrays of their respective boundary bin; these arrays are the OrBiC tables previously loaded onto the GPU. These arrays enable the kernel's threads to access the full-resolution data and corresponding row-identifier information for all records that lie in the boundary bin. Threads characterize each record as passing or failing based on comparisons made with the accessed full resolution data (lines 4, 5, 9, and 12 in Algorithm 2). The results of these logical comparisons are written to *Sol*[], ***not*** using the thread's identifier as an index, but the accessed row identifier (obtained from *rowID*) corresponding to the evaluated record.

**Compound Range Queries.** From a high level, we answer a compound range query by logically combining the solutions obtained from a sequence of simple range queries. To perform this task efficiently, we direct each simple query's kernel to utilize the same solution space in GPU memory. The compound range query's solution is produced once each simple query has been answered. In more complicated cases, e.g. "(X1 AND X2) OR (X3 AND X4)", the solution to each basic compound query can be written to a unique bit in the GPU's solution space; the bits can then be combined in each GPU kernel as needed (through bit-shifts) to form the solution to the query.

From a lower level, DP-BIS answers the first simple range with the kernels outlined in Algorithms 1 and 2. These kernels perform unconditional writes to the compound range query's solution space. More specifically, all threads "initialize" this solution

[1] Each GPU thread has a unique ID that aids in coordinating highly parallel tasks. These unique IDs form a series of continuous integers, $0 \rightarrow \mathit{maxThread}$, where maxThread is the total thread count set for the GPU kernel.

space with the first simple range query's solution. All subsequent simple range queries, however, utilize logic-based (AND, OR, etc.) derivatives of these kernels. These logic-based kernels only differ from the kernels outlined in Algorithms 1 and 2 in the final write operation each thread performs (lines 5, 9, and 12, and lines 6, 10, and 13 respectively). These changes, shown in table 2, ensure that each thread, logically combines the current simple range query's solution with the existing compound range query's solution. Section 6.2 demonstrates the implementation and performance of this approach.

## 5 Datasets, Index Strategies, and Test Setup

In this section we describe the datasets and index strategies we use in our performance analysis. We discuss testing parameters at the end of this section.

All tests were run on a desktop machine running the Windows XP operating system with SP2. All GPU kernels were run utilizing NVIDIA's CUDA software: drivers version 1.6.2, SDK version 1.1 and toolkit version 1.1. Our hardware setup consists of an Intel QX6700 quad-core multiprocessor, 4 GB of main memory, and a SATA storage system that provides 70 MB/s sustained data transfer rates. The GPU co-processor we use is NVIDIA's 8800GTX. This GPU provides 768 MB of memory and can manage over 12,000 concurrent execution threads.

### 5.1 Datasets

We use two datasets in our analysis. The first dataset is produced by a scientific simulation modeling the combustion of hydrogen gas in a fuel burner. This dataset consists of seven columns where each subsequent column increases in row size: 50 million rows for column one, 100 million rows for column two,. . . , 350 million rows for column seven. We use this dataset in Section 6.1 to measure and compare the effect that increasing column size has on processing and I/O performance.

The second dataset we use is synthetically produced. This dataset consists of 7 columns each with 50 million rows. Each column consists of a series of randomly selected, randomly distributed values from a range of floating point values [-32767.0, 32767.0]. In Section 6.2 we answer a series of compound range queries over this data. This experiment measures and compares the processing and I/O costs of finding the union or intersection between an increasing number of columns.

In both datasets, the records consist of 32-bit floating point data. The time to build the DP-BIS index for each column in our datasets is shown in Table 3; note the cost in time to build the indices scales well with the increasing size of the base data. In this table, the size for the DP-BIS index includes the size for the encoded data table, as well as the size for the OrBiC base and row identifier data tables.

### 5.2 Index Strategies

In our tests, we evaluate the I/O and processing performance of two indexing strategies: DP-BIS and the projection index. We independently evaluate the concurrency each index affords by implementing and testing the performance of a CPU-based and a GPU-based version of the index.

**Table 3.** This table shows the index sizes and DP-BIS index build times for each column used in our tests. The size for the DP-BIS index includes the size for the encoded data table, as well as the size for the OrBiC base and row identifier data tables. All times represent an average build time calculated from ten test builds.

| | *Column Size (-in millions of rows-)* | | | | | | |
|---|---|---|---|---|---|---|---|
| | *50* | *100* | *145* | *200* | *250* | *300* | *350* |
| *Base Data Size (-in MB-)* | 200 | 400 | 580 | 800 | 1000 | 1200 | 1400 |
| *DP-BIS Index Size (-in MB-)* | 450 | 900 | 1305 | 1800 | 2250 | 2700 | 3150 |
| *Index Build Time (-in minutes-)* | 1.22 | 2.65 | 4.12 | 5.82 | 7.49 | 9.37 | 11.36 |

The CPU-based DP-BIS index is implemented on a multi-core CPU that contains four CPU cores. In this implementation, the work of answering the query is divided separately and equally over each CPU core through the use of Pthreads [41]; here each CPU core is assigned an individual thread and a portion of the DP-BIS low-resolution and full-resolution data to evaluate.

The GPU-based DP-BIS index is implemented on a GPU using the constructs of the data parallel programming language CUDA. This implementation is directly based on the method presented in Section 4.4.

The CPU projection index begins by reading each full-resolution column into CPU memory space. The query is answered by simply performing comparisons on the array(s) without any additional data structure. We use this strategy in our tests because it provides a good baseline for assessing performance.

The GPU projection index is similar to the CPU projection index, with the exception that the full-resolution columns are read into GPU memory space. Additionally, all indexed values in a given column are simultaneously evaluated in parallel by the query. This indexing strategy supports the same level of concurrency offered by DP-BIS (i.e. each thread evaluates a single record), but does not provide the benefits of encoding. On the other hand, this index approach does not require performing candidate checks; a procedure that requires additional computation and read requests to GPU memory.

## 5.3 Test Setup

To ensure that all queries reflect cold-start, cold-cache behavior, we force all read operations to bypass the OS cache to prevent Windows-based data caching. Therefore, all performance times, unless otherwise stated, are based on the complete time to answer the query. This time measurement, which we refer to as the query's "total performance time", includes:

1. Disk access and data transfer times (including the cost for allocating necessary memory on the CPU and GPU),
2. time to upload data to the GPU (not applicable for the CPU-based index),
3. the time to answer a query on the uploaded data, and
4. the time to download the solution from the GPU to the CPU (again, not applicable for the CPU-based index).

In our performance analysis, we divide this total performance time into two separate time metrics, based on work-related tasks. The first time we refer to as the "I/O

**Table 4.** This table shows how the total performance time for each index strategy is composed based on I/O-related workloads, and compute-based workloads. Each value in this table represents the mean percentage of time observed for a given index strategy, based upon all tests performed in Figure 2.

| Indexing Method | *Mean Time Spent Transferring Data -as a percentage of the total time-* | *Mean Time Spent Answering the Query -as a percentage of the total time-* |
|---|---|---|
| *CPU-Projection* | 96.70 ± 0.19 | 3.30 ± 0.19 |
| *GPU-Projection* | 99.48 ± 0.26 | 0.52 ± 0.26 |
| *DP-BIS (GPU)* | 98.13 ± 1.03 | 1.87 ± 1.03 |
| *DP-BIS (CPU)* | 93.33 ± 0.7 | 6.67 ± 0.7 |

performance time". This time includes the time to perform all data transfers and memory allocation: numbers 1, 2, and 4 from the list above. The second time, which we refer to as "processing performance time", includes the time to perform all computation-related work (number 3 from the list above). In our experiments realized total, I/O, and processing performance times are recorded individually, and simultaneously. Finally, unless specified, each reported performance value represents the mean value calculated from 25 separate test runs.

## 6 Query Performance

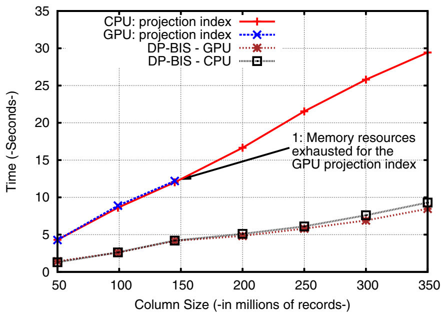
Typically, when answering a simple or compound range query over a large amount of data, far more time is spent accessing and transferring data than computing the query's solution. The performance of such I/O-intensive tasks are commonly limited by data transfer speeds. This I/O-based performance bottleneck is an especially significant challenge for multi-core architectures, like GPUs, where processing rates can far exceed bandwidth speeds [37].

We demonstrate in this section how the strategy behind DP-BIS presents one way to ameliorate this I/O-based performance bottleneck. By operating primarily on encoded data, the DP-BIS index significantly reduces the effects of this bottleneck, and uses CPU and GPU memory resources more efficiently. Additionally, the level of concurrency afforded by DP-BIS facilitates a full utilization of the thread-level parallelism provided by both multi-core CPU and GPU architectures. In this section we demonstrate the benefits of this concurrency by directly comparing processing performance times for CPU and GPU-based DP-BIS implementations. From this comparison, we show that the GPU-based implementation accelerates processing performance by a factor of 8X over the CPU-based implementation.

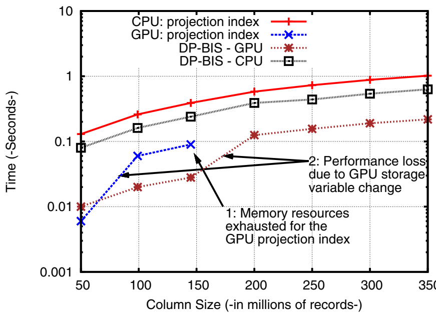
### 6.1 Answering a Simple Range Query

In our first performance evaluation, each index strategy answers a series of seven simple range queries, where each query operates on one of our scientific dataset's seven columns. In this dataset, the size of each subsequent column increases: 50 million rows, 100 million rows,..., 350 million rows.

In these experiments, we expect both CPU and GPU-based DP-BIS index to answer a simple range query using approximately 75% less time than either the CPU or GPU



(a)



(b)

**Fig. 2.** Here, (a) shows the total performance times for our three indexing strategies. In contrast (b) shows, based on the data from the same test series, *only* the processing performance time for each index. Side by side, these figures show how performance is effected by I/O plus computational workloads versus pure computational work.

projection index strategies. We base this expectation on the fact that DP-BIS primarily utilizes 8-bit low-resolution data, whereas the projection index strategies utilize 32-bit full-resolution data. We additionally expect that the GPU-based DP-BIS index will be very competitive, with respect to computational performance, with the GPU projection index. This expectation is based on the fact that both strategies support the same level of concurrency: a one-to-one mapping of threads-to-records.

**Analysis.** Figure 2(a) shows the realized total performance time of each index strategy. These performance times show that both DP-BIS implementations answer queries approximately 3X faster than both the GPU and CPU projection index. Table 4 shows how these total performance times are composed based on I/O and processing performance. Note values in Table 4 represent the average I/O and processing performance times realized for each index strategy based on the performance observed for all columns. Table 4 confirms the majority of time spent answering a simple range query is used to transfer data; each index uses over 93% of their total performance time for I/O-related tasks.

Note the GPU projection index and GPU-based DP-BIS index support the same level of concurrency when answering a simple range query. When performing this task we know both indexing strategies spend the vast majority of their time transferring data. We conclude that the disparity in total performance time experienced by the GPU projection index is directly attributable to an I/O-based performance bottleneck. This experiment illustrates the benefit of the encoding-based compression utilized by DP-BIS to accelerate the process of transferring data, and therefore the task of answering a selection query.

Aside from the performance benefits offered by DP-BIS, Figure 2(a) also highlights the benefits DP-BIS provides for GPU memory space. The GPU projection index exhausts all memory resources after columns have reached a size of 150 million rows. In comparison, DP-BIS is able to continue answering queries on columns until they reach in excess of 350 million rows. The data encoding DP-BIS utilizes thus provides over 233% better utilization of GPU memory resources when compared to the GPU projection index.

Figure 2(b) shows the processing performance times from our experiment; note that the scale of the y-axis is $\log_{10}$. Label 2 in Figure 2(b) highlights a sharp loss in performance for both the GPU-based projection index (between 50-100 million records) and DP-BIS (between 100-145 million records). This performance loss is due to a GPU implementation detail associated with how the query's solution is written for columns containing in-excess of 95 million (for the projection index) or 145 million (for DP-BIS) rows. Specifically, for columns whose row numbers exceed these values, the GPU projection index and DP-BIS can no longer store the query's solution with a 32-bit variable type (due to limited memory resources); instead an 8-bit variable type is utilized to conserve space. Writing 8-bit data to the GPU's global memory incurs significant performance penalties for both indexing strategies (as Label 2 highlights). Note however that based on the data in Table 4, this processing performance loss minimally impacts the total performance time for either of these two indexing strategies.

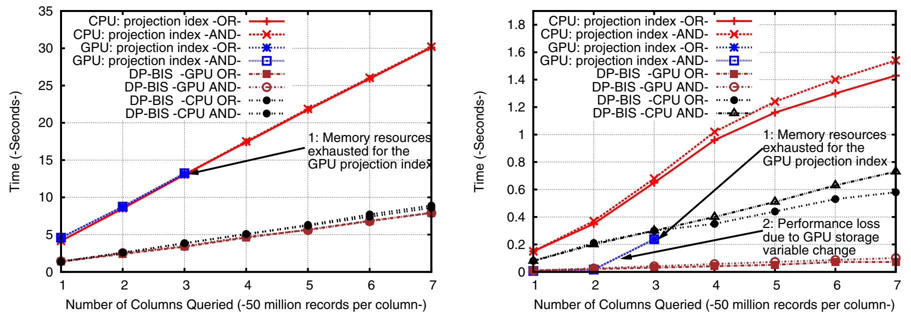
Figure 2(b) shows that before this performance loss, the GPU-based DP-BIS index answers queries up to 13X faster than the CPU projection index, 8X faster than the CPU-based DP-BIS index, and (for columns containing more than 95 million records) 3.4X faster than the GPU projection index. After this loss in performance, the GPU-based DP-BIS index outperforms the CPU-based projection and DP-BIS index by 4.9X and 3X respectively.

The concurrency afforded by DP-BIS is evident in comparing the processing performance times for the CPU-based and GPU-based implementations. The GPU-based DP-BIS index answers the query up to 8X faster than the CPU-based implementation. This acceleration in processing performance is a direct consequence of the GPU's increased thread-level parallelism over the multi-core CPU. Accelerated processing performance times are critical for many scientific applications, e.g. query-driven visualization (QDV) [38,39,40], where data can be presumed to be read once, cached by the OS, and queried repeatedly during analysis stages. In these applications, user workloads are driven more by processing performance times, which make up a larger percentage of the analysis workload, then by disk access times. For these applications, GPU-based implementations of DP-BIS provide significant performance benefits.

### 6.2 Answering a Compound Range Query

In this second performance evaluation, we use both indexing strategies to answer seven separate compound range queries. The columns our queries evaluate are taken from our synthetic dataset, where each column contains 50 million rows. In this series of tests, each subsequent query will constrain an additional column; in the final test, each index strategy will answer a query constraining seven columns. In this experiment, we perform this series of tests twice: once where we find the intersection, and once where we find the union of all columns queried. We refer to these logic-based series of tests as conjunction ($X_1 \bigwedge X_2 \bigwedge ... \bigwedge X_7$), and disjunction ($Y_1 \bigvee Y_2 \bigvee ... \bigvee Y_n$) tests.

We expect to see some level of disparity in processing performance times between the conjunction and disjunction tests. This expectation is based on the fact that, in our kernels, identifying the intersection between two records requires slightly more computational overhead than identifying their union. We additionally note on these tests that the GPU-based DP-BIS implementation will not require a variable change for the



(a) (b)

**Fig. 3.** Here, (a) shows the total performance times for our three indexing strategies when they perform a series of conjunction and disjunction tests. In contrast (b) shows, based on the data from the same test series, *only* the processing performance time for each index.

GPU's solution space. More specifically, 50 million rows is a comparatively small solution space and therefore DP-BIS will be able to utilize the more efficient 32-bit data type throughout the entire experiment. As a result, we expect DP-BIS will maintain its performance trend and not suffer the performance drop highlighted by Label 2 in Figure 2(b) from the previous experiment.

**Analysis.** Figure 3(a) shows the total performance times of all index strategies for both the conjunction and disjunction tests. In both experiments, the DP-BIS implementations answer compound range queries some 3–3.7X faster than the projection strategies. Note that Figure 3(a) shows no disparity between the conjunction and disjunction tests; such performance disparities are processing based and are more easily revealed in the processing performance times, shown in Figure 3(b).

Figure 3(b) highlights several important trends. First, as expected, the conjunction tests require more time to answer than the disjunction tests: 5–7% more time for the CPU projection index, and 20–27% more time for the CPU and GPU-based DP-BIS index. This performance trend is not readily seen in the GPU projection index; the lack of data points, due to exhausted memory resources (Label 1), obscures this performance disparity. Label 2 in Figure 3(b) highlights the loss of performance experienced by the GPU projection index due to the variable type change made in the GPU's solution space (see Section 6.1). In comparison, DP-BIS does *not* require such a change to the solution pace. Unlike the experiments performed in Section 6.1 (see Figure 2(b)), the smaller solution space employed by these tests (50 versus 350 million rows) enables DP-BIS to consistently use the more efficient 32-bit variable type. Finally, the processing performance benefits for a GPU-based implementation of DP-BIS are clearly seen in Figure 3(b); the GPU-based implementation of DP-BIS is 8X faster then the CPU-based implementation.

## 7 Conclusions

In the next decade, the evolution and predominance of multi-core architectures will significantly challenge and change the way data processing is done in the database community. As CPUs rapidly continue to become more like parallel machines, new strategies must be developed that can fully utilize the increasing thread-level parallelism, and thus the processing capabilities, of these architectures.

In presenting DP-BIS, we provide a parallel indexing data structure that will scale effectively with the future increase of processor cores on multi-core architectures. We also provide a parallelizable encoding-based compression strategy that enables DP-BIS to significantly reduce the I/O overhead associated with answering a range query.

We are currently developing a *nested* binning strategy (i.e., binning the records contained in bins) that will enable DP-BIS to provide even further processing and I/O performance benefits. Related to this work, we are additionally optimizing DP-BIS performance with the development of a two-level cache: one cache for the GPU and one for the CPU. This two-level cache will increase DP-BIS I/O performance by caching more frequently used boundary bin data in the GPU cache, and less frequently used boundary bin data in a larger CPU cache. Finally, we are integrating DP-BIS with several scientific data formats (netCDF, and HDF) to generate a new query API. This API will enable users to efficiently generate complex selections on netCDF and HDF datasets.

## Acknowledgments

This work was supported by Lawrence Berkeley National Laboratories, and by the Office of Science at the U.S. Department of Energy under Contracts No. DE-AC02-05CH11231 and DE-FC02-06-ER25777 through the Scientific Discovery through Advanced Computing (SciDAC) program's Visualization and Analytics Center for Enabling Technologies (VACET), the Institute for Ultra-Scale Visualization, and the Scientific Data Management Center.

## References

1. Becla, J., Lim, K.T.: Report from the workshop on extremely large databases (2007)
2. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM 51, 107–113 (2008)
3. Gray, J., Liu, D.T., Nieto-Santisteban, M., Szalay, A., DeWitt, D., Heber, G.: Scientific data management in the coming decade. CTWatch Quarterly (2005)
4. Asanovic, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker, W.L., Shalf, J., Williams, S.W., Yelick, K.A.: The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, Electrical Engineering and Computer Sciences, University of California at Berkeley (2006)
5. DeWitt, D., Gray, J.: Parallel database systems: the future of high performance database systems. Commun. ACM 35, 85–98 (1992)
6. Raman, R., Vishkin, U.: Parallel algorithms for database operations and a database operation for parallel algorithms. In: Proc. International Parallel Processing Symposium (1995)
7. Litwin, W., Neimat, M.A., Schneider, D.A.: LH*—a scalable, distributed data structure. ACM Trans. Database Syst. 21, 480–525 (1996)
8. Norman, M.G., Zurek, T., Thanisch, P.: Much ado about shared-nothing. SIGMOD Rec. 25, 16–21 (1996)
9. Bamha, M., Hains, G.: Frequency-adaptive join for shared nothing machines. Parallel and Distributed Computing Practices 2 (1999)

10. Rahayu, J.W., Taniar, D.: Parallel selection query processing involving index in parallel database systems. In: ISPAN 2002, p. 0309 (2002)
11. Govindaraju, N.K., Lloyd, B., Wang, W., Lin, M.C., Manocha, D.: Fast computation of database operations using graphics processors. In: Proc. of SIGMOD, pp. 215–226 (2004)
12. Govindaraju, N., Gray, J., Kumar, R., Manocha, D.: GPUTeraSort: high performance graphics co-processor sorting for large database management. In: Proc. of SIGMOD, pp. 325–336 (2006)
13. Fang, R., He, B., Lu, M., Yang, K., Govindaraju, N.K., Luo, Q., Sander, P.V.: GPUQP: query co-processing using graphics processors. In: Proc. SIGMOD, pp. 1061–1063 (2007)
14. He, B., Yang, K., Fang, R., Lu, M., Govindaraju, N., Luo, Q., Sander, P.: Relational joins on graphics processors. In: Proc. SIGMOD, pp. 511–524 (2008)
15. Sun, C., Agrawal, D., Abbadi, A.E.: Hardware acceleration for spatial selections and joins. In: Proc. of SIGMOD, pp. 455–466 (2003)
16. O'Neil, P.E., Quass, D.: Improved query performance with variant indexes. In: Proc. of SIGMOD, pp. 38–49 (1997)
17. Comer, D.: The ubiquitous B-tree. Computing Surveys 11, 121–137 (1979)
18. Gaede, V., Günther, O.: Multidimension access methods. ACM Computing Surveys 30, 170–231 (1998)
19. Wu, K., Otoo, E.J., Shoshani, A.: Optimizing bitmap indices with efficient compression. ACM Trans. on Database Systems 31, 1–38 (2006)
20. Stockinger, K., Wu, K., Shoshani, A.: Evaluation strategies for bitmap indices with binning. In: Galindo, F., Takizawa, M., Traunmüller, R. (eds.) DEXA 2004. LNCS, vol. 3180, pp. 120–129. Springer, Heidelberg (2004)
21. Antoshenkov, G.: Byte-aligned bitmap compression. In: Proc. of the Conference on Data Compression, p. 476 (1995)
22. Antoshenkov, G., Ziauddin, M.: Query processing and optimization in ORACLE RDB. In: Proc. of VLDB, pp. 229–237 (1996)
23. Wu, K., Otoo, E., Shoshani, A.: On the performance of bitmap indices for high cardinality attributes. In: Proc. of VLDB, pp. 24–35 (2004)
24. Boncz, P.A., Zukowski, M., Nes, N.: MonetDB/X100: Hyper-Pipelining Query Execution. In: Proc. Conference on Innovative Data Systems Research, Asilomar, CA, USA, pp. 225–237 (2005)
25. Stonebraker, M., Abadi, D.J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N., Zdonik, S.: C-store: a column-oriented dbms. In: Proc. of VLDB, pp. 553–564 (2005)
26. Gray, J., Liu, D.T., Nieto-Santisteban, M.A., Szalay, A.S., DeWitt, D.J., Heber, G.: Scientific data management in the coming decade. SIGMOD Record 34, 34–41 (2005)
27. Zhang, R., Ooi, B.C., Tan, K.L.: Making the pyramid technique robust to query types and workloads. In: Proc. of ICDE, p. 313 (2004)
28. O'Neil, P.E.: Model 204 architecture and performance. In: Gawlick, D., Reuter, A., Haynie, M. (eds.) HPTS 1987. LNCS, vol. 359, pp. 40–59. Springer, Heidelberg (1989)
29. Amer-Yahia, S., Johnson, T.: Optimizing queries on compressed bitmaps. In: Proc. of VLDB, pp. 329–338 (2000)
30. Wu, K., Stockinger, K., Shoshani, A.: Breaking the curse of cardinality on bitmap indexes. In: Ludäscher, B., Mamoulis, N. (eds.) SSDBM 2008. LNCS, vol. 5069, pp. 348–365. Springer, Heidelberg (2008)
31. Sinha, R.R., Winslett, M.: Multi-resolution bitmap indexes for scientific data. ACM Trans. Database Syst. 32, 16 (2007)
32. Glatter, M., Huang, J., Gao, J., Mollenhour, C.: Scalable data servers for large multivariate volume visualization. Trans. on Visualization and Computer Graphics 12, 1291–1298 (2006)

33. McCormick, P., Inman, J., Ahrens, J., Hansen, C., Roth, G.: Scout: A hardware-accelerated system for quantitatively driven visualization and analysis. In: Proc. of IEEE Visualization, pp. 171–178 (2004)
34. He, B., Govindaraju, N.K., Luo, Q., Smith, B.: Efficient gather and scatter operations on graphics processors. In: Proc. of the conference on Supercomputing, pp. 1–12 (2007)
35. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.: A survey of general-purpose computation on graphics hardware. Computer Graphics Forum 26, 80–113 (2007)
36. Lieberman, M.D., Sankaranarayanan, J., Samet, H.: A fast similarity join algorithm using graphics processing units. In: Proc. of ICDE, pp. 1111–1120 (2008)
37. NVIDIA Corporation: NVIDIA CUDA compute unified device architecture programming guide (2007), http://developer.nvidia.com/cuda
38. Bethel, E.W., Campbell, S., Dart, E., Stockinger, K., Wu, K.: Accelerating network traffic analysis using query-driven visualization. In: Proc. of the Symposium on Visual Analytics Science and Technology, pp. 115–122 (2006)
39. Stockinger, K., Shalf, J., Wu, K., Bethel, E.W.: Query-driven visualization of large data sets. In: Proc. of IEEE Visualization, pp. 167–174 (2005)
40. Gosink, L., Anderson, J.C., Bethel, E.W., Joy, K.I.: Variable interactions in query driven visualization. IEEE Trans. on Visualization and Computer Graphics. 13, 1400–1407 (2007)
41. Nichols, B., Buttlar, D., Farrell, J.P.: Pthreads Programming. O'Reilly, Sebastopol (1998)

# Finding Regions of Interest in Large Scientific Datasets

Rishi Rakesh Sinha[1], Marianne Winslett[2], and Kesheng Wu[3]

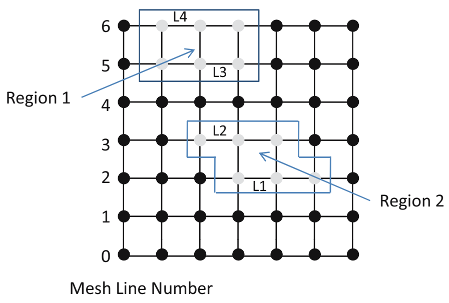
[1] Microsoft Corporation, Redmond
[2] Department of Computer Science University of Illinois at Urbana-Champaign
[3] Lawrence Berkeley National Laboratory

**Abstract.** Consider a scientific range query, such as *find all places in Africa where yesterday the temperature was over 35 degrees and it rained.* In theory, one can answer such queries by returning all geographic points that satisfy the query condition. However, in practice, users do not find this low-level answer very useful; instead they require the points to be consolidated into regions, i.e., sets of points that all satisfy the query conditions and are adjacent in the underlying mesh. In this paper, we show that when a high-quality index is used to find the points and a good traditional connected component labeling algorithm is used to build the regions, the cost of consolidating the points into regions dominates range query response time. We then show how to find query result points and consolidate them into regions in expected time that is sublinear in the number of result points. This seemingly miraculous speedup comes from a point lookup phase that uses bitmap indexes and produces a compressed bitmap as the intermediate query result, followed by a region consolidation phase that operates directly on the intermediate query result bitmap and exploits the spatial properties of the underlying mesh to greatly reduce the cost of consolidating the result points into regions. Our experiments with real-world scientific data demonstrate that in practice, our approach to region consolidation is over 10 times faster than a traditional connected component algorithm.

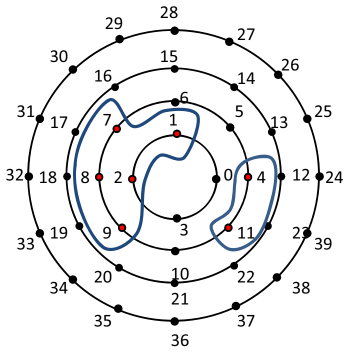
## 1 Introduction

Most scientific data have a spatial and/or temporal aspect. Observational data are recorded at specific locations and times, while simulations record evolution of scientific phenomena over time and space. Whether produced by observation or simulation, scientific data are normally discretized into points (fine-grained individual objects). In simulations, each data point is a discrete point in space for which variables (attribute values) are calculated. The intrinsic structure of an instrument determines the finest-grained observation it can take, leading to discretization of observational data too.

The points at which readings are taken or simulated do not exist in isolation; rather, they are connected to one another in a mesh. The connectivit of a mesh can be provided implicitly, as in images and video (where every pixel



(a) A 49-point structured mesh with two regions of interest



(b) A 39 point semi-structured mesh with two regions of interest

**Fig. 1.** Example meshes

is connected to its immediate neighbors in 2- or 3-space and time) or in structured meshes (where an observation is present for every combination of X, Y, Z, and/or temporal coordinates (as in Figure 1(a))). Alternatively, connectivity can be given explicitly, as is done for semi-structured and unstructured meshes.

Given a range query, i.e., a set of range restrictions on observed or simulated values, a *region of interest* is a maximal set of connected points, each of which satisfies a set of user-specified constraints on its attributes and on the simulated or real time of the observation. Regions of interest are important in scientific query processing because they provide query answers at a higher level of abstraction than is possible with points alone. For example, a telescope records images of the sky in terms of pixels. An astronomer using a telescope to study astronomical objects might start by searching for points in an image that have at least a certain level of brightness. These points by themselves are not of interest to scientists, but when grouped together to form stars, galaxies, and quasars, they assume paramount importance.

Current scientific practice is to find regions of interest in two phases [1,2]. In the *constraint application phase*, we find all points that satisfy the given constraints (e.g., temperature $\in$ [1000, 5000] and pressure $>$ 200). The time required for this phase can vary greatly, depending on whether indexes are available to help narrow the search and if available, how effective those indexes are for multidimensional queries. In the *region growing phase*, the selected points are coalesced into connected regions. The region growing phase is usually performed with a connected component labeling algorithm in $O(N)$ time, where $N$ is the number of points satisfying the query. Our experiments will show that the typical connected component labeling algorithm dominates the query response time.

An example will provide insight into the more efficient techniques for region growing that we propose in this paper. Figure 1(a) shows two regions of interest in a 49 point structured mesh. For example, this query may represent the places in Africa where the temperature was over 35 yesterday and it rained. Consider this mesh as a sequence of horizontal lines called *mesh lines*, numbered from 0 to 6. Along a single mesh line, no two non-consecutive points are directly connected. Moreover, inter-line connections are also ordered. That is, if the $k^{th}$ point on line $l$ is directly connected to the $k^{th}$ point on line $l+1$, then the $k+1^{th}$ point on line $l$ is connected to the $k+1^{th}$ point on line $l+1$. In addition, the points on line $l$ are connected to points *only* on lines $l-1$ and $l+1$. Other kinds of structured meshes have similar intrinsic orderings or other properties that we show how to exploit in this paper, to speed up the consolidation of points into regions.

The first step in the efficient lookup of points of interest is the creation of appropriate indexes. We use bitmap indexes because they are very efficient for high-dimensional scientific queries [3], and also because they can efficiently form *query lines*, a concept explained later. Before we can construct a bitmap index, we must settle on an order for enumerating all the points in the data set. Putting it another way, we must linearize the mesh into a sequence of mesh lines. In the 2D example in Figure 1(a), we choose to linearize the mesh in row-major order, so that neighboring points in a horizontal line become consecutive points in the enumeration order. Other linearization approaches such as column-major ordering, Z-ordering, or general space-filling curves could also be used. Once a mesh is linearized, we build and compress an ordinary bitmap index for each of its attributes, for use in query processing.

As explained in Section 2, we use bitmap indexes to quickly look up the mesh points that satisfy a particular query. The output from this constraint application phase is a single compressed bit vector that represents a set of *query lines*, each of which is a maximal set of directly connected points residing on a single mesh line, such that each point satisfies the query conditions. For example, Figure 1(a) shows query lines labeled $L1$ through $L4$. In the region growing phase, we use a novel labeling algorithm to efficiently find connected components, such as the regions labeled 1 and 2 in Figure 1(a). To merge the lines in Figure 1(a) into regions, we start with mesh line 0 and look for a query line residing on that mesh line, then move on to mesh line 1 and so on. In this example, the first mesh line that contains a query line is mesh line 2, which contains query line $L1$. Then we look along mesh line 3 for query lines neighboring $L1$. To do this, we examine all the query lines residing on mesh line 3 in sorted order to find the first query line that overlaps with $L1$, which happens to be $L2$ in this case. Then we unite $L1$ and $L2$ into a single region using a union-find data structure with path compression. As there are no other query lines residing on mesh lines 3 and 4, we move to mesh line 5 and find query line L3. We then move to mesh line 6 and find query line $L4$. Since $L4$ is connected to query line $L3$, we unite the query lines $L3$ and $L4$ into a single region. Regions 1 and 2 are the final result of this algorithm.

In summary, the contribution of this paper is to show how to use compressed bitmap indexes and spatial properties of scientific meshes to find regions of interest efficiently. The major steps involved in this process are:

- *Constraint Application Phase:* Use bitmap indexes to find all points satisfying the query constraints, and encode them in a compressed bit vector.
- *Region Growing Phase:*
  - *Find query lines.* Convert the result bit vector from the constraint application phase into a set of query lines.
  - *Consolidate Lines.* Use our enhanced connected-component labeling algorithm to grow query lines into regions of interest, while exploiting mesh properties to speed up the process.

In the remainder of this paper, we present related work in Section 2, including an overview of the important properties of bitmap indexes. In Section 3, we show how to convert intermediate query results into query lines. In Section 4, we explain how to use mesh ordering properties to speed up the region growing phase. We prove that our approach runs in expected time less than O($N$), where $N$ is the number of data points satisfying the user-supplied query constraints. In Section 5, we provide methods for generating interesting orders out of unstructured meshes. In Section 6, we evaluate the performance of our query approach on two large real-world data sets. The first is a semi-structured mesh from a plasma simulation, and the second is an unstructured mesh from a fluid simulation. Our experiments show a 10 fold improvement in retrieval performance, compared to traditional methods. Section 7 concludes the paper.

## 2 Bitmap Indexes

Scientific data are typically stored as a set of arrays, with one array for each scientific variable (attribute). Each point in the array corresponds either to a real world observation point or a simulation point. The same index (i.e., array offset, or coordinates) in arrays for different attributes corresponds to the same real-world or simulated point. As mentioned earlier, bitmap indexes rely on bit vectors, which require a linear ordering of the objects they index. Popular linear orderings for common types of meshes and images are listed in Table 1. Henceforth, we refer to the linearized mesh as the *mesh vector*; the mesh vector defines the object ordering used in each bit vector in the bitmap index. Figure 2 shows one possible linearization of the mesh shown in Figure 1(a).

Bitmap indexes have gained popularity for use in append- and read-only environments, including data warehousing and scientific databases [4,5,3,6,7,8,9]. A bitmap index consists of a sequence of bit vectors, one for each value in the attribute domain. Each bit vector has one bit for each object in the data set. The $i^{th}$ bit of the $k^{th}$ bit vector is set to 1 if object or point $i$ has value $k$ for that attribute, and is set to 0 otherwise. The resulting bit vectors are usually quite sparse, and so can be compressed quite effectively using variations of run-length encoding. Figure 2 shows the uncompressed and compressed bitmap indexes corresponding to Figure 1(a), using word-aligned hybrid (WAH) compression [10] with 8-bit words (discussed later).

It is easy to see how to use an uncompressed bitmap index to answer range queries. For example, to find all objects whose value for attribute $A$ lies in the range [6, 8], we retrieve the 6th, 7th, and 8th bit vectors and perform a bitwise OR operation on them. The resulting bit vector has a 1 in the $i^{th}$ position if object $i$ is in the query answer. To find all objects whose value for $A$ lies in the range [6, 8] and whose value for attribute $B$ is in [100, 500], we use bitwise OR to find the bit vectors for the objects satisfying each of the range restrictions, and then do a bitwise AND of those two vectors.

Previous work has addressed many aspects of bitmap indexing, from its original invention [5] through investigations of the best data organization and compression approaches and algorithms for operating upon the bit vectors (e.g., [3,4,10,11]), best approaches for binning and encoding data (e.g., [12,13]), and ways to reduce lookup time by introducing hierarchy into the bitmap index [14]. In this section, we will focus on the computational and compression aspects of bitmap indexes that are most important for forming regions of interest. In general, however, our method of forming regions of interest is appropriate for use with any form of compressed bitmap index.

We adopt WAH compression because previous work has shown that it outperforms other compression schemes for bitmap indexes and is more amenable to analytical analysis [10]. WAH compression combines run-length encoding and uncompressed bitmaps. A WAH compressed bitmap contains two kinds of words: *literal* and *fill* words. The word size should be chosen to give good in-memory performance on today's platforms, e.g., 32 bits. In this case, the uncompressed bitmap is divided into 31-bit segments. The most significant bit of each WAH-compressed word is used to differentiate between literal and fill words. After its leading 0, a literal word contains one segment of the uncompressed bit vector verbatim; after its leading 1, a fill word represents a sequence of contiguous segments in the uncompressed bitmap comprised entirely of 0s or entirely of 1s. The body of a fill word tells us the number of contiguous segments that are all 0s or all 1s, as well as their value (0 or 1). WAH compressed bitmaps can be ANDed and ORed together very efficiently without significantly decompressing them; in our implementation, we follow the fourth approach described in Section 5 of [3] for ORing bit vectors. This approach does not try to recompress any words that are decompressed when creating the intermediate result bit vector. For range queries, this approach is linear in the total size of the bitmap vectors involved, and the resulting bit vector is also a WAH compressed bitmap.

WAH-compressed bitmap OR and NOT operations have expected running times that are sublinear in the number of query result points. This result is important for understanding why our point retrieval and region growing approach is sublinear in the number of query result points, so we will go over it in detail below. We consider only single attribute queries in this analysis, so we do not consider AND operations; however, the experimental results presented in Section 6 show that in practice, multi-attribute queries are also sublinear in the number of points retrieved and consolidated into regions.

Let us call a fill word followed by a group of literal bits a *run*, and call the literal bits the *tail*. A run takes at most two words to represent: one fill word and one literal word, assuming that the data set has fewer than $31 \times 2^{32}$ objects

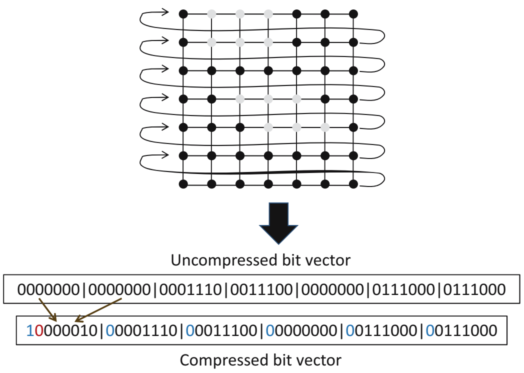


**Fig. 2.** One possible linearization of the structured mesh in Figure 1(a). The corresponding uncompressed and compressed bit vectors are shown below the mesh.

**Table 1.** Possible linearizations of different kinds of meshes

| | |
|---|---|
| Images | A simple raster ordering |
| Structured Meshes | Row-major or column-major; z-order; other space-filling curves. |
| Semi-structured Meshes | The data is typically stored in a canonical order. |
| Unstructured Meshes | linearization in the order of the node ID assigned to each node. |

in case of a 32 bit word; otherwise we can split the data set into multiple parts each having fewer than $31 \times 2^{32}$ objects. The only run that might not have a tail is the last run of a bitmap. Typically, the last few bits occupy only part of a literal word, though a whole word has to be used to store them. All together, the number of words in a WAH compressed bitmap is at most twice the number of runs. All runs except the last must contain at least one 1, else the run would not have a literal word in it. Hence each WAH-compressed bit vector has at most $n+1$ runs, where $n$ is the number of bits that are 1 in the bit vector, and has at most $2n+2$ words. Thus a set of $B$ bit vectors over $N$ objects has a maximum size of $2N+2B$. To respond to a range query over a range of $B$ bit vectors with $V$ points in the result, a maximum of of $2V+2B$ bit vectors needs to be read in memory. This analysis was first shown in [3]. In general, $V \gg B$, so the run time complexity of a range query using only ORs is $O(V)$ in the worst case.

For the negation operation, we need to change a 0 fill word to a 1 fill, and vice versa. For a literal word, we change each 0 bit to a 1 bit, and vice versa. The run time complexity of this negation operation is $O(V)$ at worst. When the range $R$ of the query is less than $\frac{|C|}{2}$, where $|C|$ is the cardinality of the domain, we apply the OR operators in the specified range. Otherwise, we apply the OR operator over the bit vectors in the complement of the specified range, and then negate the result bit vector. This reduces the overall number of bit vectors that need to be ORed to produce the result.

Let us analyze the data that leads to the worst case behavior. An occurrence of exactly one 1 in each run means that for every $31x + 30$ (where $x \geq 1$ ) zeros there is exactly one 1. This means that for each 1 there should be at least 61 zeros. Thus the cardinality of the attribute has to be at least $\frac{N}{61}$. Moreover, if we divide the mesh vector into segments of 31 points, then each value occurs at most once in two consecutive segments. Many attributes in scientific data, such as temperature and pressure, tend to be continuous in space. This means that points with similar values for the attribute tend to be clustered together in space. This continuity leads to non-trivial sized interesting regions, and is reflected as larger numbers of 1s in each run.

In the following two sections, we show how to convert intermediate result bit vectors into query lines and then grow them into regions, still in sublinear time.

## 3 From Bitmaps to Query Lines

Mesh lines are properties of a mesh and thus are static across multiple queries. A query line, on the other hand, is specific to a particular query. By definition, a query line has the following two properties:

– A query line consists of points that satisfy the query constraints, are consecutive in the mesh vector, and hence correspond to consecutive 1s in the result bit vector.
– No query line spans multiple mesh lines.

Given the set of mesh lines and the query result expressed as a WAH compressed bit vector, we can extract the query lines in a two step process that takes O($S$) time, where $S$ is the number of query lines. The two steps are:

1. Convert the compressed bit vector into a sorted sequence of line segments, where each line segment is a maximal sequence of consecutive 1s in the corresponding uncompressed bit vector. The corresponding algorithm, CreateLineSegments, is shown in Figure 3(a).
2. Split the line segments into query lines, based on the mesh lines. The corresponding algorithm, GetQueryLines, is shown in Figure 3(b).

For clarity, CreateLineSegments uses the supporting method FindNextOne(BV, position). This method returns the position of the next 1 in the uncompressed bit vector after the position passed to it. Each line segment is represented in LineSegs by its start and end point. The start point of the first line segment is the position of the first 1 in the uncompressed bit vector. CreateLineSegments sets the end of the line segment appropriately, depending on whether the current word is part of a fill word or a literal word. CreateLineSegments then iteratively finds the position of the next 1 in the uncompressed bit vector, until there are no more 1s. For each position, CreateLineSegments checks if it is the same as the end of the previous line segment. If so, it extends the previous line segment by the required number of points (depending on whether the 1 is part of a fill or a literal word). This ensures that line segments that span multiple words are correctly recorded.

```
CREATELINESEGMENTS(
BitVector BV) returns LineSegs[ ]
 i ← 0
 pos ← findNextOne(BV, 0)
 LineSegs[0].start ← pos
 if (current 1 is a start of a fill word)
   LineSegs[0].end ← pos + fillSize
 else
   LineSegs[0].end ← pos + 1
 while (BV has more bits)
 pos = findNextOne(BV, LineSegs[i])
 if (LineSegs[i].end = pos)
   if (current 1 is a start of a fill word)
      LineSegs[i].end ← pos + fillSize
   else
      LineSegs[i].end ← pos + 1
      i ← i + 1
   else
      i ← i + 1
   if (current 1 is a start of a fill word)
      LineSegs[i].start ← pos
      LineSegs[i].end ← pos + fillSize
   else
      LineSegs[i].start ← pos
      LineSegs[i-1].end ← LineSegs[i].end
+ 1
return LineSegs
```

(a) Converting a bit vector into line segments

```
GETQUERYLINES(LSegs[L],
MLines[M]) returns QLines[M][ ]
 x, y, z ← 0
 for i ← 0 to L + M
 if (LSegs[x].start > MLines[y].end
   QLines[y]   NULL
   y ← y + 1
 else if (LSegs[x].end < MLines[y].end)
   QLines[y][z].start ← LSegs[x].start
   QLines[y][z].end ← LSegs[x].end
   x ← x + 1
 else if (LSegs[x].end > MLines[y].end)
   if (LSegs[x].start _ MLines[y].start
     QLines[y][z].start ← LSegs[i].start
     QLines[y][z].end ← MLines[y].end
     z ← 0
     y ← y + 1
   else if (LSegs[x].end > MLines[x].end)
     QLines[y][z].start ← MLines[y].start
     QLines[y][z].end ← MLines[y].end
     z ← 0
     y ← y + 1
   else
     QLines[y][z].start ← MLines[y].start
     QLines[y][z].end ← LSegs[x].end
     z ← z + 1
     x ← x + 1
 return QLines
```

(b) Splitting the line segments into query lines

**Fig. 3.** Algorithm to generate query lines from result bit vectors

**Theorem 1.** CREATELINESEGMENTS *runs in $O(S)$ time, where $S$ is the number of line segments returned.*

**Proof:** First we show that CREATELINESEGMENTS extracts each individual line segment in constant time. A single line segment corresponds to a consecutive sequence of 1s in the equivalent uncompressed bit vector. Thus it can only be represented in a compressed bit vector as one of a handful of possible patterns of words. Table 2 shows these patterns with examples (using 8 bit words for simplicity). The table does not include the degenerate versions of these patterns created by chopping off first and/or last word of a pattern; these degenerate patterns can occur only at the very beginning or end of a bit vector. For example, a very short bit vector might consist only of a literal, which is a degenerate version of pattern 4.

**Table 2.** Sequences of words that can form a single query line

| | Pattern Sequence | Example |
|---|---|---|
| 1. | 0 fill - 1 fill - 0 fill | 10000011 - 11000101 - 10000101 |
| 2. | literal - 1 fill - literal | 00111111 - 11000011 - 01111110 |
| 3. | 0 fill - literal - literal - 0 fill | 10000101 - 00111111 - 01111111 - 10000011 |
| 4. | 0 fill - literal - 0 fill | 10000101 - 00111100 - 10000011 |

We show here that CreateLineSegments extracts the line segment corresponding to each of the four patterns in constant time.

1. The line segment in sequence 1 is extracted in constant time by decoding the 1 fill.
2. The line segment in sequence 2 is spread over three different words, two literals with a 1 fill in the middle. If either of the literals had $n-1$ bits set to 1 (where $n$ is the number of bits in a word), then it would have been compressed into the fill word. CreateLineSegments converts the fill word into a segment in constant time and extends the line in both directions, using the 1s in the literal words. Since the total number of 1s in the two literals is at most $2n-4$, CreateLineSegments extracts this line segment in constant time.
3. The line segment in sequence 3 is spread over two words. The maximum number of 1s in these two words is $2n-3$, as $2n-2$ consecutive 1s would have been compressed into a fill word. Thus the line segment is extracted from this pattern in constant time.
4. The line segment in sequence 4 can have at most $n-1$ 1s and hence can be extracted in constant time.

The next line segment in a WAH-compressed bit vector can either begin in the same word where the previous line segment ends, in the next word, or be separated from the previous line segment by a single 0 fill. In all three cases CreateLineSegments will take constant time to find the start of the next line segment.

We have shown that CreateLineSegments extracts each line segment in constant time and finds the beginning of the next line segment in constant time. Hence the total time to retrieve $S$ line segments using GetLineSegments is $O(S)$. □

Once GetLineSegments has found all the line segments, GetQueryLines splits them into query lines. GetQueryLines takes a sorted sequence of line segments and mesh lines as input and returns a set of query lines as output. GetQueryLines iterates over the query lines and mesh lines, splitting any line segment that spans multiple mesh lines. The query lines and the mesh lines are sorted in the same order, so that GetQueryLines can take advantage of any special mesh line orderings, as described in Section 5. The worst-case complexity of GetQueryLines is $O(L+M)$, where $L$ is the number of line segments and $M$ is the number of mesh lines. The number of mesh lines is independent of the number of points in the query result and as such from the point of view of the query run time of GetQueryLines is O($L$).

Since both GetLineSegments and GetQueryLines are O($L$), the entire process for converting the bit vector to query lines is $O(L)$. Since $L \leq S$, where $S$ is the number of query lines, we conclude that the algorithm to convert compressed bit vector query result to query lines is $O(S)$.

## 4 Region Growing Phase

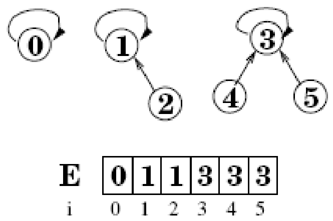
To use a connected component labeling algorithm to consolidate query lines into regions, we need to express the mesh connectivity information in a representation that efficiently supports graph operations to find the neighbors of a vertex. We use an adjacency list representation of the mesh graph, because that lets us find the neighbors of a vertex in constant time. For unstructured tetrahedral meshes, we must materialize the entire adjacency list before querying begins, while for structured meshes the neighbors can be found (computed) quickly on the fly. Materialization of the adjacency list takes time $O(E)$, where $E$ is the number of connections in the mesh. The adjacency list is materialized once per mesh and is used for all queries, so we do not include its computation time in query response time.

### 4.1 Union-Find

We make extensive use of a union-find data structure in our region growing phase [15]. The union-find data structure represents a set of disjoint sets, with each set identified by a representative element. The union operation takes two elements and computes the union of the two sets that they belong to. The find operation takes an element $e$ as input and returns the representative element of the set that $e$ belongs to. The data structure is initialized with each element contained in a separate set.

Conceptually, we represent the union-find data structure as a forest with each tree representing one disjoint set, and the root of the tree being the representative element (label) for the set. To take the union of two sets, we point the root of one tree to the other. For the find operation, we return the root of the tree that contains the element being searched for. We implement the tree using an array as our storage data structure, as shown in Figure 4(a). The smaller of the two roots is always made the root of the new united set, ensuring that $label[i] \leq i$. This constraint means that the Find algorithm moves in one direction in the array, improving cache performance. Also, by altering the label of all the points in the forest, the Label method performs path compression. With path compression, the amortized cost of $N$ Unions is O($N$) [16]. We use an array-based implementation because pointer chasing gives poor cache behavior with modern processors.

Figure 4(a) contains an example set of disjoint sets. The set is represented as a forest on the top and its array representation is shown at the bottom. Figure 4(b)



(a) The Union-Find array

| Array Index | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Initialization | 0 | 1 | 2 | 3 | 4 | 5 |
| Adding edge 1-2 | 0 | 1 | 1 | 3 | 4 | 5 |
| Adding edge 3-4 | 0 | 1 | 1 | 3 | 3 | 5 |
| Adding edge 4-5 | 0 | 1 | 1 | 3 | 3 | 3 |

(b) Operations on the Union-Find array

**Fig. 4.** Example Union-Find operations

shows the state of the array during the labeling process, from initialization to the final state. The first row shows the initialized state, while each subsequent row shows the state of the array after the addition of one edge.

## 4.2 Region Growing

The input to the region growing algorithm is a graph representation $G$ of the mesh, plus the set $L$ of query lines extracted from the result bit vector. The output of the region growing phase is a partitioning of the query lines, i.e., a set of disjoint sets $CL_i$. Each set $CL_i$ consists of the query lines that are connected in the underlying mesh and have label $i$. To compute the $CL_i$ sets, we use a special connected component labeling algorithm that is an extension of the basic connected component labeling algorithm shown in Figure 5(a). The first extension, POINTS, is a restatement of the basic algorithm for our data structures. The second extension, LINES, exploits the query lines and mesh lines to improve performance.

Figure 5(b) shows the POINTS algorithm, which uses the set $Q$ of query result points as the nodes for the union-find array. Since POINTS does not take advantage of the query lines embedded in the result of a bitmap index lookup, we treat it as the baseline algorithm in our experiments. POINTS runs in time O($E$), where $E$ is the number of edges in the forest formed by the query result points.

```
CONNECTEDCOMPONENTLABELING($G$)
    for each vertex $v \in V[G]$
        MakeSet($v$)
    for each edge($u, v$) $\in E[G]$
        if Find($u$) $\neq$ Find($v$)
            Union($u, v$)
```

(a) The basic connected component labeling algorithm

```
POINTS($G$, $Q$)
    for each point $p \in Q$
        for each neighbor $n$ of $p$ in $G$
            if $u$.Find($p$) $\neq$ $u$.Find($v$)
                $u$.Union($u$, $v$)
```

(b) The modified connected component labeling algorithm

**Fig. 5.** Connected component labeling algorithms

```
LINES($G$, $L$, $M$)
  for each line $l \in L$
      for each point $p \in l$
          $m \leftarrow$ mesh line that $p$ belongs to
          for each neighbor $n$ of $p$ that is $< m.start$
              if ($n$ is not in the mesh line of $p$)
                  if u.Find($QL[n]$) $\neq$ u.Find($QL[p]$)
                      u.Union($QL[n]$, $QL[p]$)
```

**Fig. 6.** The Lines algorithm

Figure 6 shows the LINES algorithm, which takes advantage of query lines and mesh lines to improve performance. The input to LINES is the mesh graph $G$, the query lines $L$, and the mesh lines $M$. The union-find data structure is initialized with the query lines rather than the query points. This considerably reduces the size of the data structure and hence improves performance. For each point $p$ in the current line, LINES examines each of its mesh neighbors that precede the mesh line that $p$ belongs to. LINES does not consider every neighbor of $p$, because no two query lines in the same mesh line can be directly connected. The linearization of the mesh graph allows LINES to find all query lines that belong to the same region by just checking $p$'s neighbors in lines that precede $p$'s line. This shortcut improves performance. LINES uses a hash table $QL$ to find the query line that a point belongs to. LINES is still $O(E)$ in the worst case, where $E$ is the number of edges in the forest formed by the query result points. However, as the number of points in the result grows, the length of the query lines tends to grow also, reducing the number of union operations and significantly improving performance.

## 5 Interesting Mesh Orderings

To speed up the region growing phase, LINES takes advantage of the special ordering properties of a structured mesh. Such orderings produce long mesh lines and ensure that connectivity between points in neighboring mesh lines is ordered, so that it is not necessary to look at all points in the query lines to decide whether two query lines belong to the same region. For semi-structured meshes, we have identified "canonical" mesh lines having these two properties. We have not been able to identify mesh lines with the second property for arbitrary unstructured meshes. However, we believe that such orderings do exist even in unstructured meshes, as unstructured meshes also model geometrical shapes. The properties of such shapes could lead to the creation of mesh orderings that possess both of the above properties.

We do know how to reorder the points of an unstructured mesh so that the mesh produces longer mesh lines than with its native ordering (i.e., ordered by node number). To create long mesh lines, we treat the mesh as a graph. Consider the length of the shortest path between two vertices in the graph. The

```
GLOBAL VARIABLES adjList[N][]
adjListSizes[N]
reOrder[N] $\leftarrow$ -1
counter $\leftarrow$ 0

DFS(point, line)
  for j $\leftarrow$ 0 to adjListSizes[point]
      if (reOrder[adjList[point][j]] = -1)
          if (line $\cap$ adjList[adjList[point][j]] = $\phi$)
              break;
  if (j $\neq$ adjListSizes[point])
      reOrder[adjList[point][j]] $\leftarrow$ counter
      counter $\leftarrow$ counter + 1
      line.push(adjList[point][j])
      DFS(adjList[point][j], line)

REORDERMESH()
  stack line
  for i $\leftarrow$ 0 to N
      if (reOrder[i] = -1)
          line.push(i)
          DFS(i, line)
          line.empty()
```

**Fig. 7.** Reordering the mesh using a modified DFS algorithm

maximum such length, i.e., the length of the longest shortest path, is the graph's *diameter*. The largest acyclic set of connected points lies on the diameter of the graph. Iteratively finding the diameter of the graph, removing the nodes on the diameter and finding the diameter of the remaining graph creates long mesh lines. Graph diameter computation algorithms are $O(V(E+V))$, so iteratively computing diameters of graphs is $O(V^2(E+V))$, which is too expensive for large meshes, even as a preprocessing step.

As shown in Figure 7, REORDERMESH approximates CREATELONGMESHLINES using a modified depth first search (DFS). REORDERMESH starts by picking an arbitrary point and performing a DFS from that point, changes the IDs of the points it visits to reflect the order in which they are visited. If REORDERMESH is about to visit a point that it has already visited, then it chooses a point it has not visited and starts another DFS. This repeats until REORDERMESH has visited each point in the mesh. REORDERMESH visits each edge at most once. For an unstructured mesh with 763,395 points used by the Center for Simulation of Advanced Rockets at UIUC, the native ordering of the mesh produced 505,273 mesh lines, with the longest having 3 points. With such short mesh lines, LINES offers little performance improvement over POINTS. After a run of REORDERMESH, the mesh had only 85,704 mesh lines (including many singletons). As shown later, this reordering alone makes LINES up to 10 times faster than POINTS.

## 6 Experiments

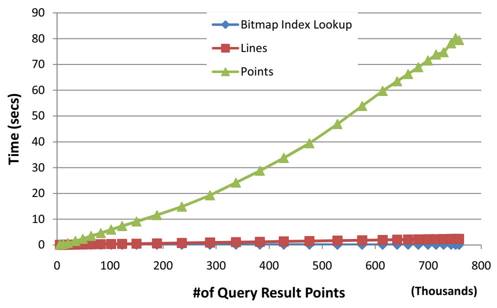
We ran our experiments on a dual core 2.4 GHz machine with 1 GB memory and two 500 GB disks, each with a manufacturer's claimed bandwidth of 100 MB/s maximum. Our mesh came from a 64 processor rocket fluid simulation at the Center for Simulation of Advanced Rockets at UIUC. The data consists of 6 different nodal variables, X, Y, Z, Vel-X, Vel-Y, Vel-Z. The data has 46 different timesteps, with 763,394 points per timestep. The mesh is an unstructured tetrahedral mesh. We constructed a bitmap index on the Z coordinate for one timestep, stored the mesh explicitly as an adjacency list, and materialized the mesh lines using ReorderMesh. We also experimented with a mesh from a plasma physics code, producing similar results that are omitted here due to space constraints. For these experiments we have used single level WAH compressed bitmaps. WAH compressed bitmap indexes have been shown to perform well for high cardinality data in [10] and hence are ideal for scientific data indexing. We do not compare our results with other indexes like R-Trees. This is because they cannot return query lines. Given that region aggregation dominates the query processing time, anything that does not return query lines would be inefficient.

**Single Variable Queries.** These experiments evaluate the difference in time for querying an unstructured mesh using Points and Lines. We ran a series of range queries on the Z coordinate. The ranges all start at $-1.6$, and end at points between $-1.4$ and 1.6, in range increments of 0.1. The results from the queries are summarized in Figure 8. The algorithms used for region growing were Points (baseline) and Lines. Figure 8(a) shows the time for bitmap index lookup (identical for Points and Lines), and the time for line segment extraction, query line extraction, and grouping into regions using Points or Lines. Lines far outperforms Points, because the degree of each point in an unstructured mesh is much higher than in a structured or semi-structured mesh, and hence Lines's reduction of the number of points to be examined improves the computation time considerably. As the number of query result points approaches the number of points in the mesh, the length of the query lines increases considerably. The choice of region growing algorithm has the greatest impact at this point, for two reasons:

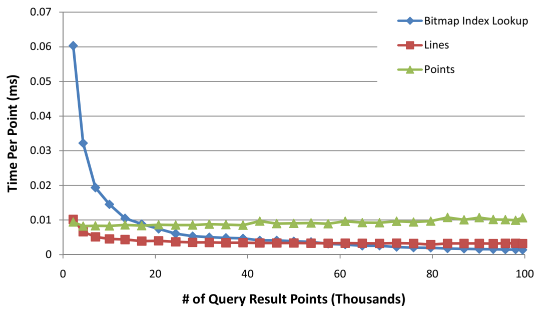
- The union-find data structure is much smaller if it is initialized with query lines rather than query points.
- The number of neighbors outside the query line decreases considerably.

We conclude that although unstructured meshes lack an ordering of the connectivity between mesh lines, we can still get significant performance benefits from the use of query and mesh lines.

The other important component in query response time, the time spent on bitmap index lookup, is quite small. In Figure 8(b), the Lines algorithm and bitmap index lookups are sublinear in the number of query result points (more clearly when the number of points are smaller). Points is clearly superlinear.



(a) Total query response time.



(b) Query response time per result point.

**Fig. 8.** Response time for range queries over the Z coordinate, with rocket simulation data

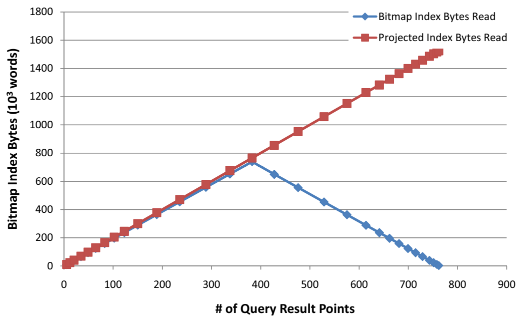


**Fig. 9.** Bitmap index bytes read into memory

Figure 9 shows a very interesting aspect of bitmap indexes. While the size of the index's bit vectors falling within the query range grows linearly with the width of the query range, we do not read all those vectors for ranges wider than half of the underlying domain. Beyond this point, the bitmap index lookup code reads the bit vectors required to answer the negation of the query, and negates the result. Hence even for very large ranges, the index lookups are inexpensive.

Figure 10 shows the performance of region retrieval algorithms for the X and Y coordinates individually. We have combined them into a single graph to show

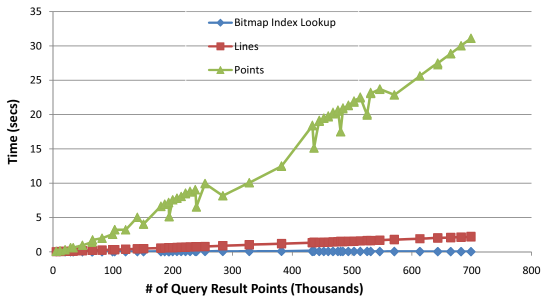
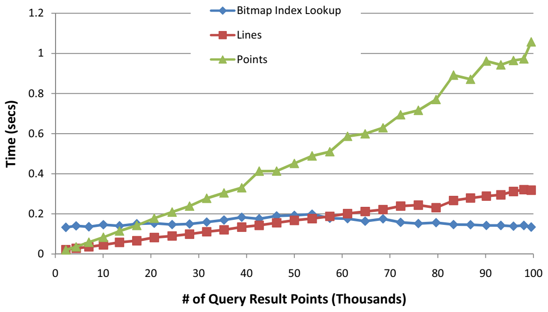
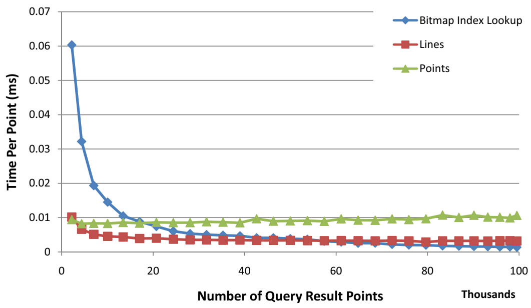


**Fig. 10.** Response time for range queries over the X and Y coordinates, with rocket simulation data



(a) Time spent over the entire query.



(b) Time spent per query point.

**Fig. 11.** Response time for range queries over X, Y, and Z attributes, for rocket simulation data

that factor on which query time depends is the number of query result points and nothing else. As before, the green, brown, and blue lines show the time taken for the region growing phase using Points, Lines, and the bitmap index lookup, respectively. Again, Lines is much faster than Points. If we look closely we can discern two distinct curves in the Points results. This is because of the presence of values from both the X and the Y coordinate query results. Such behavior is also present in the Lines algorithm and the index lookup, but is masked due to the scale of the graph.

**Multiple Variable Queries.** This set of experiments evaluates the performance of queries over multiple attributes. As the number of attributes increases, the bitmap index lookup time will also increase. We investigate the question of whether it will become the dominant factor in query performance, making the improvement in performance of the labeling algorithm superfluous.

We keep the ranges in the Z coordinates the same as in the previous experiments, and add the ranges [0, 1.2], and [-.4, .4] for the X and Y coordinates, respectively, for each query. As shown in Figure 11, when the number of points returned by the query is low, the bitmap index lookup time dominates the query response time. As the number of points increases, Points quickly becomes the major cost, while the cost of Lines increases much more slowly. Figure 11 shows the rate of growth of the query response time as a function of the number of points. Figure 11 shows that both the index lookup and Lines are sublinear in the number of result points, while Points is superlinear.

## 7 Conclusion

Previous work has shown that bitmap indexes are very effective for range queries over scientific data. In this paper, we have shown that bitmap indexes have an additional advantage: they help us to consolidate query result points into query result regions very quickly. For even larger performance gains, we can exploit our understanding of the connectivity properties of the underlying mesh, whether it is structured, semi-structured, or unstructured. We have shown both analytically and experimentally that when used together, these two techniques allow us to find query result points and consolidate them into regions in expected time that is sublinear in the number of query result points. Our new approach to region consolidation is over ten times faster than using a traditional connected component labeling algorithm for region consolidation, which is typically the dominant factor in query response time.

We have extended the techniques presented in this paper to work with semi-structured meshes (see http://dais.cs.uiuc.edu/r̃sinha/thesis_skt), by exploiting the mesh line ordering constraints found in semi-structured meshes to improve region consolidation performance significantly.

## References

1. Shi, Q., Jaja, J.F.: Efficient techniques for range search queries on earth science data. In: SSDBM, pp. 142–151 (2002)
2. Wu, K., Koegler, W., Chen, J., Shoshani, A.: Using bitmap index for interactive exploration of large datasets. In: SSDBM (2003)
3. Wu, K., Otoo, E.J., Shoshani, A.: On the performance of bitmap indices for high cardinality attributes. In: VLDB 2004, pp. 24–35 (2004)

4. Apaydin, T., Canahuate, G., Ferhatosmanoglu, H., Tosun, A.S.: Approximate encoding for direct access and query processing over compressed bitmaps. In: VLDB 2006, pp. 846–857 (2006)
5. O'Neil, P.: Model 204 architecture and performance. In: Gawlick, D., Reuter, A., Haynie, M. (eds.) HPTS 1987. LNCS, vol. 359, pp. 40–59. Springer, Heidelberg (1989)
6. Stockinger, K., Wu, K., Shoshani, A.: Evaluation strategies for bitmap indices with binning. In: Galindo, F., Takizawa, M., Traunmüller, R. (eds.) DEXA 2004. LNCS, vol. 3180, pp. 120–129. Springer, Heidelberg (2004)
7. Stockinger, K.: Design and implementation of bitmap indices for scientific data. In: IDEAS 2001, pp. 47–57 (2001)
8. Wu, M.C.: Query optimization for selections using bitmaps. In: SIGMOD 1999, pp. 227–238 (1999)
9. Wu, M.C., Buchmann, A.P.: Encoded bitmap indexing for data warehouses. In: ICDE 1998, pp. 220–230 (1998)
10. Wu, K., Otoo, E.J., Shoshani, A.: Optimizing bitmap indices with efficient compression. ACM TODS 31(1), 1–38 (2006)
11. Koudas, N.: Space efficient bitmap indexing. In: CIKM 2000, pp. 194–201 (2000)
12. Chan, C.Y., Ioannidis, Y.E.: Bitmap index design and evaluation. In: SIGMOD 1998, pp. 355–366 (1998)
13. Rotem, D., Stockinger, K., Wu, K.: Optimizing candidate check costs for bitmap indices. In: CIKM 2005, pp. 648–655 (2005)
14. Sinha, R.R., Winslett, M.: Multi-resolution bitmap indexes for scientific data. ACM TODS 32(3), 46–84 (2007)
15. Wu, K., Otoo, E., Suzuki, K.: Two strategies to speed up connected component labeling algorithms. Pattern Analysis and Applications (2008)
16. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: 21. Data Structures for Disjoint Sets. In: Introduction to Algorithms, 2nd edn., pp. 498–524. McGraw Hill, New York

# Adaptive Physical Design for Curated Archives

Tanu Malik[1], Xiaodan Wang[2], Debabrata Dash[3],
Amitabh Chaudhary[4], Anastasia Ailamaki[5], and Randal Burns[2]

[1] Purdue University, USA
tmalik@purdue.edu
[2] Johns Hopkins University, USA
{xwang,randal}@cs.jhu.edu
[3] Carnegie Mellon University, USA
ddash@cs.cmu.edu
[4] University of Notre Dame, USA
achaudha@cse.nd.edu
[5] Swiss Federal Institutes of Technology, Switzerland
anastasia.ailamaki@epfl.ch

**Abstract.** We introduce AdaptPD, an automated physical design tool that improves database performance by continuously monitoring changes in the workload and adapting the physical design to suit the incoming workload. Current physical design tools are offline and require specification of a representative workload. AdaptPD is "always on" and incorporates online algorithms which profile the incoming workload to calculate the relative benefit of transitioning to an alternative design. Efficient query and transition cost estimation modules allow AdaptPD to quickly decide between various design configurations. We evaluate AdaptPD with the SkyServer Astronomy database using queries submitted by SkyServer's users. Experiments show that AdaptPD adapts to changes in the workload, improves query performance substantially over offline tools, and introduces minor computational overhead.

## 1 Introduction

Automated physical design tools are vital for large-scale databases to ensure optimal performance. Major database vendors such as Microsoft, IBM, and Oracle now include tuning and design advisers as part of their commercial offerings. The goal is to reduce a DBMS' total cost of ownership by automating physical design tuning and providing DBAs with useful recommendations about the physical design of their databases. However, current tools [1,2,3,1] provide limited automation; they take an offline approach to physical design and leave several significant decisions during the tuning process to DBAs. Specifically, DBAs need to explicitly specify representative workloads for the tuning tool. DBAs are also required to know when a tuning session is needed and guesstimate the relative benefit of implementing the recommendations.

Complete automation is a critical requirement of libraries which will soon become data centers for curation of large scientific data. A notable example is the Sloan Digital Sky Survey (SDSS) [4] project whose data will soon be curated. The project receives a diverse workload, which exceeds a million queries every month. As such, finding a representative workload is challenging because query access patterns exhibit considerable evolution within a week [5].

The straightforward approach of running an offline tool after each query or invoking it periodically for continuous evaluation of physical design achieves most of the automation objectives. However, this approach requires further tuning by a DBA to ensure that the tool does not react too quickly or slowly and result in poor design choices. Recent research [6,7,8] on the online physical design problem focuses on index design. Bruno and Chaudhari [8] infer costs and plan properties during the query optimization phase to efficiently decide between various index configurations in an online fashion. In this paper, we focus on vertical partitioning, which is complementary to index selection. In SDSS and other scientific databases, vertical partitioning is often used because it does not replicate data, thereby reducing space requirements [1].

**Contributions.** To provide complete automation, we model the physical design problem in AdaptPD as an online problem and develop algorithms that minimize the combined cost of query execution and the cost of transitioning between configurations. We also develop efficient and accurate cost estimation modules that reduce the overhead of AdaptPD. AdaptPD is evaluated within the Astronomy database of SDSS. Experiments indicate up to two fold improvement in query response time when compared with offline tuning techniques.

We develop online algorithms that search the space of physical design alternatives without making assumptions about the workload. Analysis shows that the algorithm provides a minimum level of guarantee of adapting to workload changes. Current tools provide such guarantees only for two configurations.

Our algorithms assume a general transition model in which transition costs between configurations are asymmetric and positive. This is in contrast to current works for index design, which assume a constant cost of creating a physical design structure and zero cost of deleting them [8]. We validate our model through experiments and show that transition costs are asymmetrical and the asymmetry is bounded by a constant factor.

We develop a novel "cache-and-reuse" technique for query cost estimation. The technique caches distinct query plans that do not change across several configurations and reuses the plans for estimating query costs. By reusing cached plans, the technique minimizes computationally-intensive optimizer invocations by as much as 90%. Current tools, both offline and online, employ no such methods for query estimation and are therefore much slower to run. We also develop the first-known technique, based on bulk-inserts, for estimating the cost of transitioning from one configuration to another. In current online tools, transition costs are either fixed or assigned arbitrarily.

Our online vertical partitioning techniques have applicability beyond the automation of curated relational databases. For example, our algorithm for the regrouping of columns can also provide automation for column-store databases [9]. In particular, the algorithm is independent of whether the database is implemented as a row-store or a column-store. The techniques also have applicability in schema design of proxy database caches. We recently showed that inefficiencies in the physical design of cached objects offsets some of the benefits of deploying a cache and automated physical design can recoup that loss [10].

## 2 Related Work

Automated physical design tools use cost estimates from the optimizer or analytical I/O cost-based models to evaluate attribute groupings [1,2,3] for a given query workload. These solutions are offline, i.e., they assume a priori knowledge of the entire workload stream and therefore provide a single, static physical design for the entire workload.

Current research [7,8] emphasizes the need for automated design tools that are *always-on* and, as new queries arrive, continuously adapt the physical design to changes in the workload [11]. Quiet [6] describes an incremental adaptive algorithm for index selection which is not fully integrated with the optimizer. In Colt [7], Schnaitter et al. present a similar algorithm which relies heavily on the optimizer for cost estimation. Both approaches do not take into account transition costs. Bruno et al. present a formal approach to online index selection [8] that takes into account transition costs. Their algorithms are limited to choosing among configurations in which the only difference is the set of indices being used. Our core algorithm is general purpose in that physical design decisions are not limited to index selection. In this paper, the system is developed for configurations that are vertical partitions. We also assume that transition costs are asymmetric which is not the case in [8].

Our formulation is similar to that of task systems introduced by Borodin et al. [12]. Task systems have been researched extensively, particularly when the transition costs form a metric [12]. Our costs are not symmetric and do not form a metric. This asymmetry in transition costs exists because the sequence of operations (i.e. insertion or deletion of tables or columns) required for making physical design changes in a database exhibit different costs. The Work-Function algorithm [13] is an online algorithm for such asymmetrical task systems, but it is impractical with respect to the efficiency goals of AdaptPD. The algorithm solves a dynamic program with each query that takes $\theta(N^2)$ time, even in the best case, in which $N$ is the number of configurations. In AdaptPD we present a simpler algorithm that takes $O(N)$ time at each step in the worst case.

Read-optimized column-stores have been used for commercial workloads with considerable success [9,14]. They perform better than row-stores by storing compressed columns contiguously on disk. Column-stores, however, pose several hurdles for SDDS implementation. The implementation is well-optimized for commercial row-store databases on existing workloads and a complete migration to column-store is prohibitively expensive. Moreover, it consists of mostly floating point data that are not compressible using the RLE and bitmap compression schemes used by column-stores, thereby eliminating a crucial advantage of column-store. Our solution is an intermediate step at the storage-layer that performs workload-based regrouping of columns on a row-store and avoids increased tuple reconstruction cost associated with a column-store.

Our query cost estimation module is similar to other configuration parametric query optimization techniques, such as INUM [15], and C-PQO [16]. These techniques exploit the fact that the plan of a query across several configurations is an invariant and can be reused to reduce optimizer calls. These techniques reuse the plans when configurations are limited to sets of indices on tables. We extend plan reuse to configurations that correspond to vertical partitions.

## 3 The AdaptPD Tool

The AdaptPD tool automates several tasks of a DBA. The DBA often performs the following tasks to maintain a workload-responsive physical design: a) Identifies when workload characteristics have changed significantly such that the current physical design is no longer optimal. b) Chooses a new physical design such that excessive costs are not incurred in moving from the current physical design, relative to the benefit. The AdaptPD tool performs these tasks in an integrated fashion by continuously monitoring the workload at the granularity of a query; DBAs often monitor at the granularity of thousands of queries. It uses cost-benefit analysis to decide if the current physical design is no longer efficient and a change is required. The tool consists of three components : the core algorithm behind adaptive physical design (Section 4), a cost estimator (Section 5), and a configuration manager (Section 6).

The core algorithm solves an online problem in which the objective is to adaptively transition between different database *configurations* in order to minimize the total costs in processing a given query sequence. Given a data model, let $D = \{o_1, \ldots, o_n\}$ be the set of all possible *physical design structures* that can be constructed, which includes vertical partitions of tables, materialized views, and indices[1]. A database instance is a combination of physical design structures subject to a storage size constraint $T$ and is termed as a *configuration*. Let $\mathcal{S} = \{S_1, \ldots, S_N\}$ be the set of all possible configurations on $D$. The cost of processing a query $q$ in a configuration $S_i$ is denoted $q(S_i)$ (if $q$ cannot be processed in $S_i$ we set $q(S_i) = \infty$). Often it is necessary to change configurations to reduce query processing costs. The cost for *transitioning* between any two given configurations is given by the function $d : \mathcal{S} \times \mathcal{S} \rightarrow \Re^+$. $d$ is any function that satisfies the following properties:

1. $d(S_i, S_j) \geq 0, \forall i \neq j, S_i, S_j \in \mathcal{S}$ (positivity);
2. $d(S_i, S_i) = 0, \forall i \in \mathcal{S}$ (reflexivity); and
3. $d(S_i, S_j) + d(S_j, S_k) \geq d(S_i, S_k), \forall S_i, S_j, S_k \in \mathcal{S}$ (triangle inequality)

In particular, $d$ does not satisfy the symmetry property, *i.e.*,$\exists S_i, S_j \in \mathcal{S} d(S_i, S_j) \neq d(S_j, S_i)$. Asymmetry exists because the sequence of operations (i.e. insertion or deletion) required for making physical design changes exhibit different costs.

[1] In [11], physical design structures are referred to as *access paths*.

Given $\sigma = q_1, \ldots, q_n$, a finite sequence of queries, the objective in AdaptPD is to obtain a sequence of configurations $S = (S_0, S_1, ..., S_n), S_i \in \mathcal{S}$ such that the total cost of $\sigma$ under $S$ is minimized. The total cost is defined as

$$cost(\sigma, S) = \sum_{i=1}^{n} q_i(S_i) + \sum_{i=0}^{n-1} d(S_i, S_{i+1}), \tag{1}$$

in which the first term is the sum of costs of each query in $\sigma$ under the corresponding configuration and the second term is the total cost to transition between configurations in S. Note, if $S_{i+1} = S_i$ there is no real change in the configuration schedule. An offline optimal algorithm OPT knows the entire $\sigma$ and obtains a configuration schedule $S$ with the minimum cost. An online algorithm ALG for AdaptPD determines $S = (S_0, ..., S_n)$ without seeing the complete workload $\sigma = (q_1, ..., q_n)$. Thus, ALG determines each configuration, $S_i$, based on the workload $(q_1, ..., q_i)$ seen so far.

In this paper we focus on configurations that arise from different *vertical partitions* in the data model [1]. Let $R = \{R_1, \ldots, R_k\}$ be the given set of relations in the data model. Each configuration $S \in \mathcal{S}$ now consists of a set of *fragments* $F = \{F_1, \ldots, F_N\}$ that satisfies the following two conditions: (1) every fragment $F_i$ consists of an identifier column and a subset of attributes of a relation $R_j \in R$; and (2) each attribute of every relation $R_j$ is contained in exactly one fragment $F_i \in F$, except for the primary key.

## 4 Algorithms in AdaptPD

In this section we describe two online algorithms for the AdaptPD tool: OnlinePD and HeuPD. OnlinePD provides a minimum level of performance guarantee and makes no assumptions about the incoming workload. HeuPD is greedy and adapts quickly to changes in the incoming workload.

### 4.1 OnlinePD

We present OnlinePD, which achieves a minimum level of performance for any workload. In particular, we show its cost is always at most $8(N-1)\rho$ times that of the optimal algorithm, where $N$ is the total number of configurations in the set $\mathcal{S}$ and $\rho$ is the *asymmetry constant* of $\mathcal{S}$. Further, to achieve this performance, OnlinePD does not need to be trained with a representative workload. OnlinePD is an amalgamation of algorithms for two online sub-problems: (1) the on-line ski rental problem and (2) the online physical design problem in which the cost function $d(\cdot)$ is symmetrical. We first describe the sub-problems.

**Related Problems.** Online ski rental is a classical rent-or-buy problem. A skier, who does not own skis, needs to decide before every skiing trip that she makes whether she should rent skis for the trip or buy them. If she decides to buy skis, she will not have to rent for this or any future trips. Unfortunately, she does not know how many ski trips she will make in future, if any. This lack of

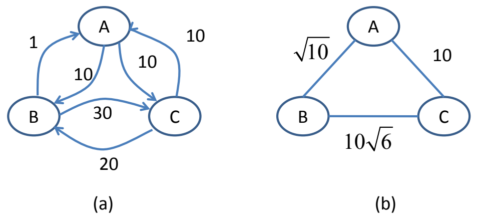


**Fig. 1.** Example of conversion from asymmetric transition costs to symmetric costs

knowledge about the future is a defining characteristic of on-line problems [17]. A well known on-line algorithm for this problem is rent skis as long as the total paid in rental costs does not match or exceed the purchase cost. Irrespective of the number of future trips, the cost incurred by this online algorithm is at most twice that of the optimal offline algorithm.

If there were only two configurations and the cost function $d(\cdot)$ satisfies symmetry, the OnlinePD problem will be nearly identical to online ski rental. Staying in the current configuration corresponds to renting skis and transitioning to another configuration corresponds to buying skis. Since the algorithm can start in any state, this leads to an algorithm that cost no more than four times the optimal.

In larger number of configurations, the key issue in establishing a correspondence with the online ski rental problem is in deciding which configuration to compare with the current one. When the costs are symmetrical, Borodin et. al [12] use *components* instead of configurations to perform an online ski rental. In particular, their algorithm recursively traverses one component until the query execution cost incurred in that component is approximately that of moving to the other component. A decision is then made to move to the other component (traversing it recursively) before returning to the first component and so on. To identify the components, they consider a complete, undirected graph $G(V, E)$ on $\mathcal{S}$ in which $V$ represents the set of all configurations, $E$ represents the transitions, and the edge weights are the transition costs. By fixing a minimum spanning tree (MST) on $G$, components are recursively determined by pick the maximum weight edge in the MST and removing it. This partitions all the configurations into two smaller components and the MST into two smaller trees.

This algorithm is shown to be $8(N-1)$-competitive [12]. ALG is $\alpha$-competitive if there exists a constant $b$ such that for every finite query sequence $\sigma$,

$$cost(\texttt{ALG}\ on\ \sigma) \leq \alpha * cost(\texttt{OPT}\ on\ \sigma) + b. \tag{2}$$

OPT is the offline optimal that has complete knowledge of $\sigma$. OnlinePD extends the above algorithm to solve the problem in which costs are asymmetrical. It does so by *transforming* its complete, directed graph on $\mathcal{S}$ and $d(\cdot)$ into a complete, undirected graph and applying any algorithm for online physical design in which costs are symmetrical. We describe the transformation and use Borodin's algorithm to show that it increases cost at most $8(N-1)\rho$ times of OPT.

**Transformation in OnlinePD.** When there are only two configurations, a simple transformation in which graph edges are replaced by the *sum* of transition costs gives a 3-competitive algorithm [10]. However, adding transition costs provides poor bounds for an $N$ node graph. To achieve better competitive performance, we transform the directed graph into a undirected graph as follows:

Let $G'$ be the directed graph. In $G'$, replace every pair of directed edges $(u, v)$ and $(v, u)$ with an undirected edge $(u, v)$ and a corresponding transition cost equal to $\sqrt{d(u,v).d(v,u)}$ irrespective of the direction. This transforms $G'$ into $H$. $H$ has the following two properties because of the transformation: a) If $p$ is a path in $H$ and $p'$ is the corresponding path in $G'$ (in any one direction), then $\frac{cost(p)}{\sqrt{\rho}} \leq cost(p') \leq \sqrt{\rho} cost(p)$. The inequality allows us to bound the error introduced by using $H$ instead of $G'$. b) $H$ violates the triangle inequality constraint. This is shown by a simple three-node example in Figure 1(a). In this example, a three node directed, fully connected graph with $\rho = 10$ is transformed to an undirected graph in Figure 1(b). The resulting triangle does not obey triangle inequality. OnlinePD exploits the fact that Borodin's algorithm constructs an MST, which makes it resilient to the triangle inequality violation.

Algorithm 1 details OnlinePD in which Algorithm 2 is a subroutine. To construct the traversal before processing queries, the MST is built on a graph in which edge weights are rounded to the next highest power of two. Let the maximum rounded weight in the MST, denoted by $F$ in the Algorithm 1, be $2^M$. We establish the proof using $F$.

```
Input: Directed Graph: $G(V, E_o)$ with weights $d(\cdot)$, Query Sequence: $\sigma$
Output: Vertex Sequence to process $\sigma$: $u_0, u_1, \ldots$
Transform $G$ to undirected graph $H(V, E)$ s.t. $\forall (u, v) \in E$ weight
$d_H(u, v) \leftarrow \sqrt{d(u, v) \cdot d(v, u)}$;
Let $B(V, E)$ be the graph $H$ modified s.t. $\forall (u, v) \in E$ weight
$d_B(u, v) \leftarrow d_H(u, v)$ rounded to next highest power of 2;
Let $F$ be a minimum spanning tree on $B$;
$\mathcal{T} \leftarrow \mathsf{traversal}(F); u \leftarrow S_0$;
while there is a query $q$ to process do
    $c \leftarrow q(u)$;
    Let $v$ be the node after $u$ in $\mathcal{T}$;
    while $c \geq d_B(u, v)$ do
        $c \leftarrow c - d_B(u, v); u \leftarrow v$;
        $v \leftarrow$ the node after $v$ in $\mathcal{T}$;
    end
    Process $q$ in $u$;
end
```

**Algorithm 1**: OnlinePD($G$)

**Lemma 1.** *Any edge in $\mathcal{T}$ of rounded weight $2^m$ is traversed exactly $2^{M-m}$ times in each direction.*

*Proof.* We prove by induction on the number of edges in $F$. For the base case, there are no edges in $F$, and the lemma is trivially true. For the inductive case,

```
Input: Tree: $F(V,E)$
Output: Traversal for $F$: $\mathcal{T}$
if $E = \{\}$ then
    $\mathcal{T} \leftarrow \{\}$;
else if $E = \{(u,v)\}$ then
    Return $\mathcal{T}$: Start at $u$, traverse to $v$, traverse back to $u$;
else
    Let $(u,v)$ be a maximum weight edge in $E$, with weight $2^M$;
    On removing $(u,v)$ let the resulting trees be $F_1(V_1,E_1)$ and $F_2(V_2,E_2)$,
    where $u \in V_1$, and $v \in V_2$;
    Let maximum weight edges in $E_1$ and $E_2$ have weights $2^{M_1}$ and $2^{M_2}$
    respectively; $\mathcal{T}_1 \leftarrow \mathsf{traversal}(F_1)$;
    $\mathcal{T}_2 \leftarrow \mathsf{traversal}(F_2)$;
    Return $\mathcal{T}$: Start at $u$, follow $\mathcal{T}_1$ $2^{M-M_1}$ times, traverse $(u,v)$, follow $\mathcal{T}_2$
    $2^{M-M_2}$ times;
end
```

**Algorithm 2**: $\mathsf{traversal}(F)$

let $(u,v)$ be the maximum weight edge in $F$ used in $\mathsf{traversal}(\cdot)$, and similarly let $F_1$ and $F_2$ be the trees obtained by removing $(u,v)$. Now the edge $(u,v)$ is traversed exactly once in each direction as required by the lemma. By the inductive hypothesis, each edge of $F_1$ of rounded weight $2^m$ is traversed exactly $2^{M_1-m}$ times in each direction in the traversal $\mathcal{T}_1$, in which $M_1$ is the maximum rounded weight in $F_1$. Since $\mathcal{T}$ includes exactly $2^{M-M_1}$ traversals of $\mathcal{T}_1$, it follows that each such edge is traversed $2^{M-m}$ times in each direction in $\mathcal{T}$. The same reasoning applies to edges in $F_2$.

**Theorem 1** *Algorithm* OnlinePD *is* $4(N-1)(\rho+\sqrt{\rho})$*-competitive for the* OnlinePD *problem with* $N$ *configurations and asymmetry constant* $\rho$*.*

*Proof.* During each traversal of $F$, the following two statements are true: (i) the cost of OnlinePD is at most $2(N-1)2^M(1+\sqrt{\rho})$, and (ii) the cost of the offline optimal is at least $2^{M-1}/\sqrt{\rho}$. The theorem will then follow as the cost of OnlinePD during any single traversal is constant with respect to the length of $\sigma$. We prove (i) following Lemma 1 and (ii) from induction. (See proof in Appendix).

The bound of $8(N-1)\rho$ in OnlinePDis only a worst case bound. In our experiments, OnlinePD performs much better than best known offline algorithms for this problem and tracks closely with the workload adaptive algorithm HeuPD.

## 4.2 HeuPD

HeuPD chooses between neighboring configurations greedily. The current configuration in HeuPD ranks its neighboring configurations based on the estimated query execution costs in the neighboring configurations. HeuPD keeps track of the cumulative penalty of remaining in the current configuration relative to every other neighboring configuration for each incoming query. A transition is made once HeuPD observes that the benefit of a new configuration exceeds a threshold. The threshold is defined as the sum of the costs of the most recent transition and next transition. HeuPD is described in detail in [10] and presented here as an alternative algorithm in AdaptPD. AdaptPD combines HeuPD with cost-estimation procedures described in this paper.

Let $x \in \mathcal{S}$ be the current configuration and $y \in \mathcal{S}$ be the neighboring configuration in which $y \neq x$. Define $\delta^y_{\max}(k)$ as the maximum cumulative penalty of remaining in $x$ rather than transitioning to $y$ at query $q_k$ (the penalty of remaining in $x$ for $q_k$ is $q_k(x) - q_k(y)$). In HeuPD, this transition threshold is a function of the configuration immediately prior to $x$ and the alternative configuration being considered. Let $z$ be the configuration immediately prior to $x$ in which the threshold required for transitioning to $y$ is $d(z, x) + d(x, y)$. The decision to transition is greedy; that is, HeuPD transitions to the *first* configuration $y$ that satisfies $\delta^y_{\max}(k) > d(z, x) + d(x, y)$.

## 5 Cost Estimation in AdaptPD

OnlinePD and HeuPD require $O(NlogN)$ time and space for pre-processing and $O(N)$ processing time per query. In this section we describe techniques to reduce $N$. Physical design tools also incur significant overhead in query cost estimation. Transition costs are often assigned arbitrarily, providing no correlation between the costs and actual time required to make transitions. Thus, we describe techniques for accurate and efficient cost estimation for vertical partitioning.

### 5.1 Transition Cost Estimation

We present an analytical transition model that estimates the cost of transitions between configurations. In an actual transition, data is first copied out of a database and then copied into new tables according to the specification of the new configuration. Gray and Heber [18] recently experimented with several data loading operations in which they observed that SQL bulk commands such as `BULK_INSERT` command in SQL Server work much like standard bulk copy (bcp) tools but are far more efficient than bcp as it runs inside the database. We base our analytical model on performance results obtained from using `BULK_INSERT` on a 300 column table.

We observe two artifacts of the `BULK_INSERT` operation. First, copying data into the database is far more expensive than copying data out of the database. Second, cost of importing the data scales linearly with the amount of data being copied into the database. The first artifact is because data is normally copied out in native format but is loaded into the database with type conversions and key constraints. The linear scaling is true because `BULK_INSERT` operations mostly incur sequential IO. We model the cost of importing a partition $P$:

$$BCP(P) = cR_P W_P + kR_P \tag{3}$$

in which $R_P$ is the number of rows, $W_P$ is the sum of column widths, $c$ is the per byte cost of copying data into the database, and $k$ is the per row cost of constructing the primary key index. Thus, the estimated cost is the sum of importing data and the cost of creating a primary key index. Index creation cost is linear due to a constant overhead with each new insert. Constants $c$ and $k$ are system dependent and can be easily determined using regression on few "sample" `BULK_INSERT` operations. In this model, we assume no cost for transaction logging, which is disabled for fast copy.

The transition cost model uses Equation 3 to model the cost of moving from a configuration $S_i$ to another configuration $S_j$. Let configuration $S_i$ consists of partitions {T1$_i$,...,Tm$_i$}, $S_j$ consists of partitions {T1$_j$,...,Tn$_j$} and $\Delta_{ij}$ be the partition set difference {T1$_j$,...,Tn$_j$} - {T1$_i$,...,Tm$_i$}, The transition cost is:

$$d(S_i, S_j) = \sum_{t \in \Delta_{ij}} BCP(t) \tag{4}$$

## 5.2 Query Cost Estimation

We present an efficient and yet accurate technique for estimating query costs across several configurations. The technique is based on the idea that cached query plans can be reused for query cost estimation. The traditional approach of asking the optimizer for the cost of each query on each configuration is well-known to be very expensive [15]. By caching and reusing query plans, the technique avoids invoking the optimizer for cost estimation and achieves an order of magnitude improvement in efficiency. To maintain high accuracy, the technique relies on recent observations that the plan of a query across several configurations is an invariant. By correctly determining the right plan to reuse and estimating its cost, the technique achieves the complementary goals of accuracy and efficiency. We describe conditions under which plans remain invariant across configurations and therefore can be cached for reuse. We then describe methods to cache the plans efficiently and methods to estimate costs on cached plans.

**Plan Invariance.** We illustrate with an example when the plan remains invariant across configurations and when it does not. Figure 2 shows three different configurations $S_1, S_2, S_3$ on two tables `T1(a,b,c,d)` and `T2(e,f,g,h)` with primary and join keys as `a` and `e`, respectively. Consider a query $q$ that has predicate clauses on $c$ and $d$ and a join clause on $a$ and $e$: `select T1.b, T2.f from T1, T2 where T1.a = T2.e and T1.c > 10 and T1.d = 0`. Let the query be optimized in $S_1$ with the shown join methods and join orders. The same plan is optimal in $S_2$ and can be reused. This is because $S_2$'s partitions with respect to columns $c$ and $d$ are identical to $S_1$'s partitions. In $S_3$, however, the plan cannot be used as columns $c$ and $d$ are now merged into a single partition. This is also reflected by the optimizer's choice which actually comes with a different plan involving different join methods and join orders.

Plan invariance can be guaranteed if the optimizer chooses to construct the same plan across different configurations.

**Theorem 1.** *The optimizer constructs the same query plan across two configurations $S_1$ and $S_2$, if the following three conditions are met:*

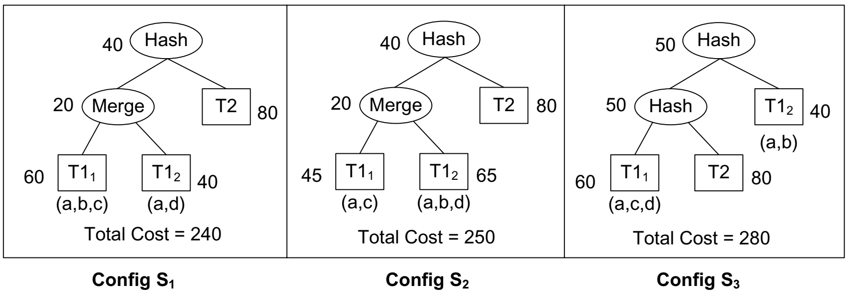


**Fig. 2.** Plan reuse across configurations. (Estimated costs are for illustration only).

1. *Configurations have the same number of partitions with respect to all columns mentioned in the query.*
2. *The division of the predicate columns in $S_1$ and $S_2$ is exactly the same.*
3. *If $\delta(S_1)$ and $\delta(S_2)$ define the page size distributions of $S_1$ and $S_2$ respectively, then dist$(\delta(S_1), \delta(S_2)) < \epsilon$. Where dist function determines the distance between two page distributions, and $\epsilon$ is a DBMS dependent constant.*

Condition 1 guarantees the same number of joins in the plan for any two configurations. For instance, if $S_2$ partitions `T1` into three partitions, then the optimizer joins twice instead of once to reconstruct the rows of the original table. Since the resulting plan is different from $S_1$, plan reuse cannot be guaranteed. If the query does not select on $b$, then the same plan can still be reused.

Condition 2 guarantees similar cardinality of the intermediate join results so that the optimizer selects the same join order and method to find the optimal plan. Condition 2 is illustrated in Figure 2 in which keeping $c$ and $d$ in different partitions leads to a merge joins in $S_1$ and $S_2$. A hash join is preferred in $S_3$ when $c$ and $d$ are grouped together.

Condition 3 avoids comparing drastically different configurations in terms of page distribution. The *dist* function can be a standard distribution distance, such as KL-divergence [19], and $\epsilon$ can be determined by experimenting over large number of plans. That is if a large table, with say 100 columns, has two configurations and if in the first configuration partitions are of uniform sizes, (*i.e.* two partitions with each partition containing 50 columns) and in the second configuration partitions are highly skewed (*i.e.* one partition has one column and the second has all the remaining columns), then the optimizer does not construct the same plan. In particular, the optimizer prefers to join equi-sized partition tables earlier and delays joining skewed tables as long as possible. Hence reusing the plan of one configuration for the other provides inaccurate results.

If above three conditions are satisfied, we prove by contradiction that the optimizer generates the same plan for any given two configurations. Suppose the join method and join order for $S_i$ is $J_1$ and for $S_j$ is $J_2$. By our assumption, $J_1$ and $J_2$ are different. Without loss of generality, let $J_1$ costs less than $J_2$ if we ignore the costs of scanning partitions. Since the configurations $S_i$ and $S_j$ have the same orders (primary key orders for all partitions), select the same number of rows from the partitions, and the page size of the filtered rows are similar, $J_1$ can still be used for $S_j$. Since using $J_1$ reduces the total cost for running the query on $S_j$, it implies that $J_2$ is not the optimal plan, which contradicts our assumption that $J_2$ is part of the optimal plan.

**The Plan Cache.** We cache the query plan tree with its corresponding join methods, join-orders, and partition scans. The stripped plan tree is uniquely identified by $<query\_id, partition\_list, page\_distribution>$. The first part of the string identifies the query for which the plan is cached, the second part specifies the list of partitions in which columns in the predicate clause of the query occur, and the third part specifies the page distribution of each partition.

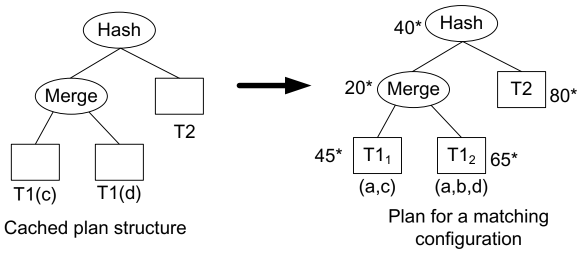


**Fig. 3.** The plan for $S_1$ cached with the key $[query - id, T1((c),(d)), T1(128,64)]$ on the left. On the right is the estimated plan for the new configuration $S_2$.

**Cost Estimation.** Cost estimation involves accurate estimation of partition scan costs and join costs. Thus given a new configuration, we first retrieve its corresponding plan from the cache using the key and then estimate the costs of partition scans and join methods. Partition scan costs are estimated by computing the average cost of scanning the partition in the cached plan and multiplying it with actual size of partition in the new configuration. Thus if $c$ is the I/O cost of the scanning operation in the cached plan, and $s_0$ is the size of the vertical partition in the cached plan, the cost of partition scan in the new configuration is estimated as $f = c \times \frac{s}{s_0}$. In this $s$ is the size of the new partition. To estimate the cost for joining partitions using the join methods from the cache, we adopt the System-R's cost model, developed by Selinger et al. [20]. The System-R cost model gives us an upper bound on the actual join costs, and according to our experiments predicts the plan cost with 98.7% accuracy on average.

## 6 Experiments

We implement our online partitioning algorithms and cost estimation techniques in the SDSS [4] Astronomy database. We describe the experimental setup before presenting our main results. This includes analysis of workload evolution over time, performance of various online and offline algorithms, and accuracy of cost estimation.

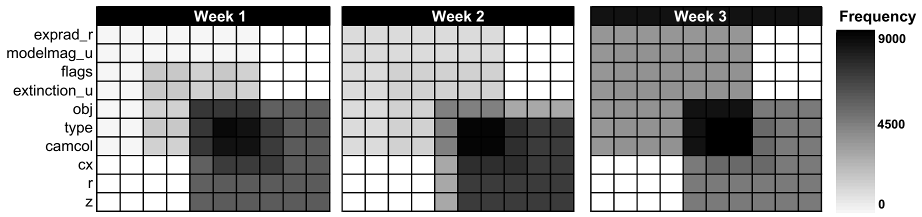


**Fig. 4.** Affinity matrix (co-access frequency) for ten select attributes from the *PhotoObjAll* table

## 6.1 Experimental Setup

**Workload Characteristics.** We use a month-long trace from SDSS consisting of 1.4 million read-only queries. The queries consist of both simple queries on single tables and complex queries joining multiple tables. Also, queries are template-based [5] and can be summarized compactly using templates. However, considerable *evolution* occurs in the workload in that new templates are introduced continually and prior templates may disappear entirely.

Figure 4 captures workload evolution for the first three weeks of the trace. It shows the affinity matrix for ten attributes from a single table in which each grid entry corresponds to the frequency with which a pair of attributes are accessed together (ordering of attributes are the same along the row and column). The basic premise is that columns that occur together and have similar frequencies should be grouped together in the same relation [21]. The results show that column groupings change on a weekly basis. An online physical design tool which continuously monitors the workload can evaluate whether transitioning to a new configuration will lead to a improvement in overall cost.

**Comparison Methods.** We contrast the performance of OnlinePD with several online and offline algorithms. OnlinePD has polynomial-time complexity and finds the minimal spanning tree using the Prim's algorithm. While it makes no assumptions about the workload, this generality comes at a cost. Namely, given domain specific knowledge about the workload, highly tuned workload adaptive algorithms can be designed. To measure the cost of generality, we compare OnlinePD with HeuPD (Section 4.2). We also compare against `AutoPart`, an existing, offline vertical partitioning algorithm. `AutoPart` is presented with the entire workload as input during initialization. This incurs an initial overhead to produce a physical design layout for the workload, but it can service the workload with no further tuning. `AutoPartPD` is another physical design strategy that employs the offline `AutoPart` algorithm. Unlike `AutoPart`, it is adaptive by running `AutoPart` daily (incurs one transition at the beginning of each day) and it is prescient in that the workload for each day is provided as input *a priori*. Finally, `NoPart` serves as the base case in which no vertical partitioning is used.

**Costs.** The transition costs are estimated using the analytical model each time a new template is introduced. The estimates show that the asymmetry constant

(Section 4) $\rho$ is bounded and its maximum value is approximately 3.25. The model itself estimates the transition costs with 87% accuracy in which transition costs are considered accurate if they are within 10% of the actual costs. Query cost estimation is done using the cache and reuse technique, which provides 94% accuracy in our experiments.

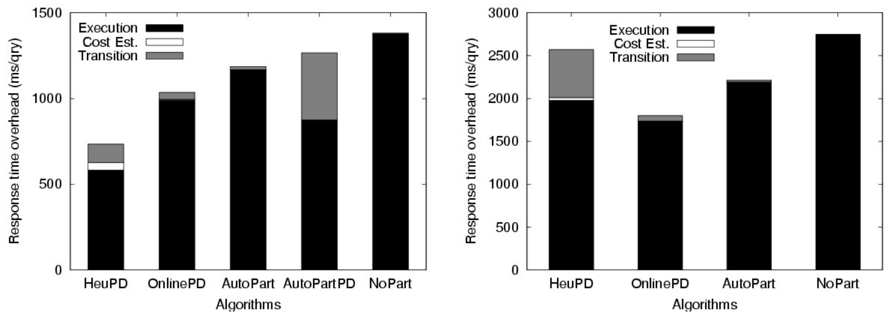
**Database.** For I/O experiments, we execute queries against a five percent sample (roughly 100GB in size) of the DR4 database. Although sampling the database is less than ideal, it is necessary to finish I/O experiments in a reasonable time for real workloads. Given the time constraints, we compromised database size in order to accommodate a larger workload, which captures workload evolution over a longer period. To sample the database, we first sample the fact table consisting of all celestial objects (*PhotoObjAll*) and then sample the remaining tables through foreign key constraints.

The data is stored in Microsoft's SQL Server 2000 on a 3GHz Pentium IV workstation with 2GB of main memory and two SATA disks (a separate disk is assigned for logging to ensure sequential I/O). Microsoft SQL Server does not allow for queries that join on more than 255 physical tables. This is required in extreme cases in which the algorithm partitions each column in a logical relation into separate tables. Hammer and Namir [22] show that between the two configurations with each column stored separately or all columns stored together, the preferred configuration is always the latter. In practice, this configuration does not arise because the cost of joining across 255 tables is so prohibitive that our algorithm never selects this configuration. To reduce the configuration space, we do not partition tables that are less than 5% of the database size. This leads to a large reduction in the number of configurations with negligible impact on performance. The total number of configurations is around 5000.

**Performance Criteria.** We measure the cost of algorithms in terms of average query response time. This is the measure from the time a query is submitted until the results are returned. If a transition to a new configuration is necessary, the algorithm undergoes a transition before executing the query. This increases the response time of the current query but amortizes the benefit over future queries. Our results reflect average response time over the entire workload.

## 6.2 Results

We compute the query performance by measuring its response time on the proxy cache using the sampled database. Figure 5(a) provides the division of response time for query execution, cost estimation using the optimizer, and transitions between configurations. (The total response time is averaged over all queries). OnlinePD improves on the performance of `NoPart` by a factor of 1.5 with an average query execution time of 991 ms. Not surprisingly, HeuPD, which is tuned specifically for SDSS workloads, further improves performance by 40% and exhibits two times speedup over `NoPart`. This improvement is low considering that OnlinePD is general and makes no assumptions regarding workload access patterns. `NoPart` suffers due to higher scan costs associated with reading extraneous



(a) SDSS Workload (b) Adversarial Workload

**Fig. 5.** Distribution of response time overhead

columns from disk. Likewise, `AutoPart` suffers by treating the entire workload as an unordered set of queries and providing a single, static configuration during initialization. Even `AutoPartPD` did not improve response time beyond the offline solution because the benefits of periodic physical design tuning is offset by high, daily transition costs. Thus, adapting offline solutions such as `AutoPart` to evolving workloads is challenging because they do not continuously monitor for workload changes nor account for transition cost in tuning decisions.

Another interesting feature of the results is that OnlinePD incurs much lower transition costs than HeuPD. This artifact is due to the conservative nature of OnlinePD. It evaluates only two alternatives at a time and transitions only if it expects significant performance advantages. On the other hand, HeuPD responds quicker to workload changes by evaluating all candidate configurations simultaneously and choosing a configuration that benefits the most recent sequence of queries. This optimism of HeuPD is tolerable in this workload but can account for significant transition costs in workloads that change more rapidly relative to SDSS. To appreciate the generality of OnlinePD over a heuristic solution, we evaluated a synthetic SDSS workload that is adversarial with respect to HeuPD in Figure 5(b). In particular, the workload is volatile and exhibits no stable state in the access pattern, which causes HeuPD to make frequent, non-beneficial transitions. As a result, Figure 5(b) shows that OnlinePD exhibits a lower query execution time and a factor of 1.4 improvement over HeuPD.

Figure 5(a) also shows the average response time of performing cost estimation (time spent querying the optimizer). For `AutoPart`, this is a one-time cost incurred during initialization. In contrast, cost estimation is an incremental overhead in OnlinePD and HeuPD. HeuPD incurs a ten folds overhead in cost estimation over OnlinePD (43 ms versus 4 ms). This is because HeuPD incurs 93 calls to the optimizer per query. Thus, HeuPD benefits immensely from QCE due to the large number of configurations that it evaluates for each query. Reusing cached query plans allow HeuPD to reduce cost estimation overhead by ten folds and avoid 91% of calls to the optimizer. Without QCE, the total average response time of HeuPD is 1150 ms, which would lag the response time of OnlinePD by 4 ms. As such, HeuPD scales poorly as the number of alternative

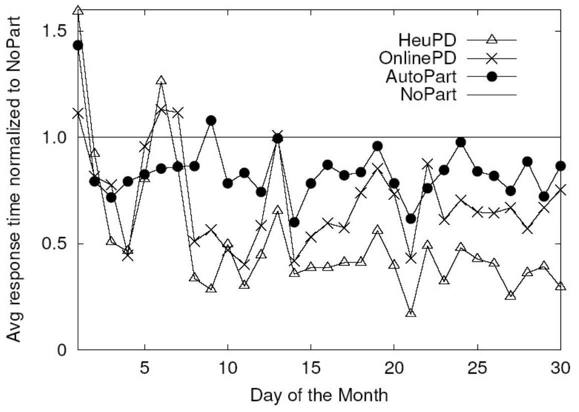


**Fig. 6.** Average daily response time overhead normalized to NoPart

configurations increases. This make OnlinePD attractive for proxy caches which receive a continuous stream of queries and decisions have to be made rapidly.

Finally, we refer to the average transition cost (cost of changing between configurations) from Figure 5(a). `AutoPart` only incurs a single transition during initialization while `NoPart` incurs no transition cost. `AutoPartPD` incurs the highest overhead, requiring a complete reorganization of the database on a daily basis. HeuPD makes 768 minor configuration changes compared with 92 for OnlinePD which leads to a three times per query overhead in transition cost (113 ms compared with 43 ms). Thus, while OnlinePD is slower than HeuPD at detecting and adapting to changes in the workload, it benefits with fewer transitions that disrupt the execution of incoming queries.

Figure 6 charts the average daily response time (both query execution and transition cost) for various algorithms normalized to `NoPart`. There is significant fluctuations in average response times resulting from workload changes over time. While all algorithms improve on `NoPart`, `AutoPart` tracks most closely with `NoPart` since neither implements changes to the physical design after initialization. OnlinePD and HeuPD further improves response time, but exhibits several performance spikes (most notably on days one, six, and thirteen) that perform no better than `NoPart`. These indicate significant workload changes that cause more transitions to occur that delay completion of certain queries. The transition overhead is greatest for OnlinePD and HeuPD on day one and remains more stable afterward because at initialization, all tables are unpartitioned.

Figure 7 shows the cumulative distribution function (CDF) of the error in cost estimation using QCE instead of the optimizer. This error is determined by:

$$abs\left(1-\left(\frac{\texttt{QCE est. query cost}}{\texttt{Optimizer est. query cost}}\right)\right) \tag{5}$$

Consider the dashed-line in the plot, which corresponds to the errors in cost estimation for all queries. Although the average cost estimation error is only 1.3%, the plot shows that the maximum error in cost estimation is about 46%,

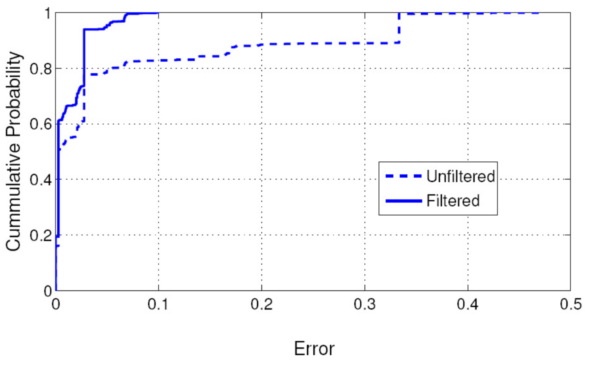


**Fig. 7.** Error in cost estimation for QCE. Dashed line represents all queries and solid line represents queries with higher than 5 unit cost.

with about 14% of the estimations with more than 10% error. Inspecting high error estimations reveals that the errors occur in queries with estimated costs below 5 optimizer cost units. We plot the CDF for errors after removing those light queries. The solid line in Figure 7 shows the cost estimation error for these filtered set of queries. The maximum error for the filtered queries is about 11%, and about 94% of the estimations have less than 5% error.

The inaccuracies in the light queries comes from the approximations discussed in Section 5.2. Since the contribution of light queries to workload cost is insignificant compared to the heavy queries (4% of our workload), the inaccuracy in estimating their costs does not affect the configurations selected by our algorithm.

## 7 Summary and Future Work

In this paper, we have presented AdaptPD, a workload adaptive physical design tool that automates some of the DBA tasks such as estimating when to tune the current physical design and finding representative workloads to feed the physical design tool. The tool quantitatively compares the current configuration with other possible configurations, giving the DBA a good justification of the usefulness of the recommended design. Automation of such tasks reduces the cost of ownership of large database systems such as the SDSS in which physical design tuning is routinely performed by DBAs. Since these tools gradually change ownership from DBAs to curators, it is essential to minimize the overhead of administration and yet ensure good performance.

We have developed novel online techniques that adapt to drastic changes in the workload without sacrificing the generality of the solution. The techniques are supported by efficient cost estimation modules that make them practical for continuous evaluation. Experimental results for the online algorithm show significant performance improvement over existing offline methods and tracks closely with heuristic solution tuned specifically for SDSS workloads. These tuning tools are not specific to vertical partitions and can be extended to index design, which is our primary focus going forward.

## References

1. Papadomanolakis, S., Ailamaki, A.: AutoPart: Automating Schema Design for Large Scientific Databases Using Data Partitioning. In: SSDBM (2004)
2. Agrawal, S., Narasayya, V.R., Yang, B.: Integrating Vertical and Horizontal Partitioning Into Automated Physical Database Design. In: SIGMOD (2004)
3. Chu, W.W., Ieong, I.T.: A Transaction-Based Approach to Vertical Partitioning for Relational Database Systems. IEEE Trans. Software Eng. 19(8), 804–812 (1993)
4. The Sloan Digital Sky Survey, http://www.sdss.org
5. Wang, X., Malik, T., Burns, R., Papadomanolakis, S., Ailamaki, A.: A Workload-Driven Unit of Cache Replacement for Mid-Tier Database Caching. In: Kotagiri, R., Radha Krishna, P., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 374–385. Springer, Heidelberg (2007)
6. Sattler, K.U., Geist, I., Schallehn, E.: QUIET: Continuous Query-Driven Index Tuning. In: VLDB (2003)
7. Schnaitter, K., Abiteboul, S., Milo, T., Polyzotis, N.: COLT: Continuous On-line Tuning. In: SIGMOD (2006)
8. Bruno, N., Chaudhuri, S.: An Online Approach to Physical Design Tuning. In: ICDE (2007)
9. Stonebraker, M., Abadi, D., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N., Zdonik, S.: C-Store: A Column Oriented DBMS. In: VLDB (2005)
10. Malik, T., Wang, X., Burns, R., Dash, D., Ailamaki, A.: Automated Physical Design in Database Caches. In: SMDB (2008)
11. Agrawal, S., Chu, E., Narasayya, V.: Automatic Physical Design Tuning: Workload as a Sequence. In: SIGMOD (2006)
12. Borodin, A., Linial, N., Saks, M.E.: An Optimal Online Algorithm for Metrical Task System. J. ACM 39(4), 745–763 (1992)
13. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for server problems. J. Algorithms 11(2), 208–230 (1990)
14. Abadi, D., Madden, S., Hachem, N.: Column-Stores Vs. Row-Stores: How Different Are They Really. In: SIGMOD (2008)
15. Papadomanolakis, S., Dash, D., Ailamaki, A.: Efficient Use of the Query Optimizer for Automated Database Design. In: VLDB (2007)
16. Bruno, N., Nehme, R.: Configuration Parametric Query Optimization for Physical Design Tuning. In: SIGMOD (2008)
17. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York (1998)
18. Heber, G., Gray, J.: Supporting Finite Element Analysis with a Relational Database Backend Part II: Database Design and Access. Technical Report, Microsoft Research (2006), http://research.microsoft.com/apps/pubs/default.aspx?id=64571
19. Kullback, S., Leibler, R.A.: On information and sufficiency. Ann. Math. Statistics (1951)
20. Selinger, P., Astrahan, M., Chamberlin, D., Lorie, R., Price, T.: Access Path Selection in a Relational Database Management System. In: SIGMOD (1979)
21. Navathe, S., Ceri, S., Wiederhold, G., Dou, J.: Vertical Partitioning Algorithms for Database Design. ACM Trans. Database Syst. 9(4), 680–710 (1984)
22. Hammer, M., Niamir, B.: A Heuristic Approach to Attribute Partitioning. In: SIGMOD (1979)

## Appendix: Competitiveness of OnlinePD

We prove that OnlinePD algorithm is $O(4\rho(N-1))$ competitive. Reusing the notations from Section 4, in Algorithm 1 we construct the graph $B$ by rounding up the cost of edges on the undirected transition graph to a power of two. We build an MST on $B$ and call it $F$. We then build a traversal on $F$ using Algorithm 2 and denote that traversal as $\mathcal{T}$. Let the maximum rounded weight in the tree $F$ be $2^M$. The following proof is inspired by the proof in [12]

**Lemma 2.** *If the maximum edge weight in $\mathcal{T}$ is $2^M$, any edge in $\mathcal{T}$ of rounded weight $2^m$ is traversed exactly $2^{M-m}$ times in each direction.*

*Proof.* We prove by induction on the number of edges in $F$. For the base case, there are no edges in $F$, and the lemma is trivially true. For the inductive case, let $(u, v)$ be the maximum weight edge in $F$ used in the $\mathsf{traversal}(\cdot)$, and similarly let $F_1$ and $F_2$ be the trees obtained by removing $(u, v)$. Now the edge $(u, v)$ is traversed exactly once in each direction as required by the lemma. By the inductive hypothesis, each edge of $F_1$ of rounded weight $2^m$ is traversed exactly $2^{M_1-m}$ times in each direction in the traversal $\mathcal{T}_1$, in which $M_1$ is the maximum rounded weight in $F_1$. Since $\mathcal{T}$ includes exactly $2^{M-M_1}$ traversals of $\mathcal{T}_1$, it follows that each such edge is traversed $2^{M-m}$ times in each direction in $\mathcal{T}$. Exactly the same reasoning applies for edges in $F_2$.

**Theorem 2** *Algorithm* OPDA *is* $4(N-1)(\rho+\sqrt{\rho})$*-competitive for the* OnlinePD *problem with $N$ configurations and asymmetry constant $\rho$.*

*Proof.* We shall prove that during each traversal of $F$: (i) the cost of OPDA is at most $2(N-1)2^M(1+\sqrt{\rho})$, and (ii) the cost of the offline optimal is at least $2^{M-1}/\sqrt{\rho}$. The theorem will then follow as the cost of OPDA during any single traversal is constant with respect to the length of $\sigma$.

To prove (i), recall from Lemma 2 that any edge in $\mathcal{T}$ of rounded weight $2^m$ is traversed exactly $2^{M-m}$ times. Thus the total rounded weight traversed for an edge is $2 \cdot 2^{M-m} \cdot 2^m = 2 \cdot 2^M$. By construction of the algorithm the total processing cost incurred during $\mathcal{T}$ at a node just before a traversal of this edge is $2 \cdot 2^M$. The total transition cost incurred during $\mathcal{T}$ in a traversal of this edge is at most $2 \cdot 2^M \sqrt{\rho}$, since the cost $d(\cdot)$ can be at most $\sqrt{\rho}$ times larger than the corresponding $d_B(\cdot)$. This proves (i) as there are exactly $N-1$ such edges.

We prove (ii) by induction on the number of edges in $F$. Suppose $F$ has at least one edge, and $(u, v)$, $F_1$, and $F_2$ are as defined in $\mathsf{traversal}(\cdot)$. If during a cycle of $\mathcal{T}$, OPT moves from a vertex in $F_1$ to a vertex in $F_2$, then since $F$ is a minimum spanning tree, there is no path connecting $F_1$ to $F_2$ with a total weight smaller than $d_B(u, v)/(2\sqrt{\rho}) = 2^{M-1}/\sqrt{\rho}$. Otherwise during the cycle of $\mathcal{T}$, OPT only stays in one of $F_1$ or $F_2$; w.l.o.g. assume $F_1$. If $F_1$ consists of just one node $u$, and OPT stays there throughout the cycle of $\mathcal{T}$, then by definition of the algorithm, OPT incurs a cost of at least $d_B(u, v) = 2^M \geq 2^{M-1}/\sqrt{\rho}$. If $F_1$ consists of more than one node, then by the induction hypothesis, OPT incurs a cost of at least $2^{M_1-1}/\sqrt{\rho}$ per cycle of $T_1$. Since during one cycle of $\mathcal{T}$ there are $2^{M-M_1}$ cycles of $\mathcal{T}_1$, OPT incurs a cost of at least $2^{M-1}/\sqrt{\rho}$. This completes the proof.

# MLR-Index: An Index Structure for Fast and Scalable Similarity Search in High Dimensions*

Rahul Malik, Sangkyum Kim, Xin Jin, Chandrasekar Ramachandran, Jiawei Han, Indranil Gupta, and Klara Nahrstedt

University of Illinois at Urbana-Champaign, Urbana IL 61801, USA
{rmalik4,kim71,xinjin3,cramach2,hanj,indy,klara}@illinois.edu

**Abstract.** High-dimensional indexing has been very popularly used for performing similarity search over various data types such as multimedia (audio/image/video) databases, document collections, time-series data, sensor data and scientific databases. Because of the *curse of dimensionality*, it is already known that well-known data structures like kd-tree, R-tree, and M-tree suffer in their performance over high-dimensional data space which is inferior to a brute-force approach *linear scan*. In this paper, we focus on an approximate nearest neighbor search for two different types of queries: *r-Range search* and *k-NN search*. Adapting a novel concept of a *ring* structure, we define a new index structure *MLR-Index* (**M**ulti-**L**ayer **R**ing-based **Index**) in a metric space and propose time and space efficient algorithms with high accuracy. Evaluations through comprehensive experiments comparing with the best-known high-dimensional indexing method *LSH* show that our approach is faster for a similar accuracy, and shows higher accuracy for a similar response time than *LSH*.

## 1 Introduction

A similarity search finds a small set of objects near a given query. Here, similarity between objects is often measured by their features, which are points in a high-dimensional vector space. Efficient similarity searches have become more and more important in various domains such as multimedia (audio/image/video), web documents, time-series, sensor and scientific areas because of the rapid mounting of these datasets and the increasing desires of modern users for fast and scalable systems.

Usually, feature vectors of these complex datasets lie in a high-dimensional space which are quite often bigger than several hundred dimensions. But for sufficiently high dimensions, existing index structures degrade to a linear scan approach which compares a query to each point of the database, that is not

* The work was supported in part by the U.S. NSF grants CNS-0520182, IIS-0513678, IIS-0842769, CAREER CNS-0448246, ITR CMS-0427089 and Air Force Office of Scientific Research MURI award FA9550-08-1-0265. Any opinions, findings, and conclusions expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

scalable [1]. Fortunately, in many cases, it is not necessary to insist on exact answers [2], and recent research has focused on this direction: searching approximate nearest neighbors (or *ANN*) [3,4,5,6]. But even for this relaxed problem, *ANN*, there exist fundamental challenges to achieve time and space efficiency with high accuracy performance.

An interesting approach *VQ-Index* [3] achieved an efficient compression by using a vector quantization technique which resulted in a smaller search space. Quantization was conducted by partitioning the space into regions and assigning a representative to each region so that data points were mapped to the representative of the region they fall into. One problem is that it is still had to perform a linear scan over the representatives repeatedly to retrieve similar representatives and then again over the candidates from the region of the resulted similar representatives, which made the searching procedure inefficient.

The best-known solution *LSH* [4,6] for *ANN* problem shows fast response time for a given query, but it needs to use multiple (sometimes over hundreds of) hash tables to achieve high accuracy, while a linear scan takes linear time but does not require any additional memory usage. Our object is to find a solution which shows a similar or better time efficiency to *LSH* using less memory.

In this paper, based on the vector quantization idea, but instead of conducting a linear scan over the representatives of the regions, we utilize a novel index structure *MLR-Index* (**M**ulti-**L**ayer **R**ing-based **Index**) which adapts a ring structure first introduced in a distributed network context [7]. Each representative (named *search node*) keeps track of $O(\log N)$ peers and organizes them into a set of concentric rings centered on itself with exponentially increasing radii. This multi-layer ring structure provides not only the immediate vicinity information but also outer pointers to remote regions. Instead of linear performance of simple vector quantization, it can find the nearest search node in $O(\log N)$ steps. Here, $N$ is the number of search nodes not the number of all data objects. In our experiments, *MLR-Index* based similarity search algorithms showed several times faster response time than *LSH* for similar accuracy performance.

The followings are the contributions of this paper.

– We design a novel index structure *MLR-Index* that fits in a high-dimensional space with quantized vectors.
– We propose approximate algorithms which are time and space efficient with high accuracy for two different nearest neighbor search problems. We also propose even faster methods by using an additional data structure with similar accuracy.
– We develop scalable hierarchical search algorithms to maximize performances for dense regions which are clustered in multi-levels.
– We perform extensive experiments on real datasets comparing with the *state-of-the-art* high-dimensional indexing algorithm *LSH* [4,6] which show the superior performance of *MLR-Index.*

The remainder of the paper is structured as follows. Section 2 describes related works on high-dimensional similarity search. Section 3 defines terms and problems of this paper, and introduces concepts of the search node and its multi-layer ring structure. Based on these concepts, an efficient method for finding nearest search node is described in Sect. 4. Section 5 proposes the basic similarity search algorithms. Section 6 describes more efficient algorithms using $m$-NS structure. In Sect. 7, a hierarchical search method for a dense region is proposed to utilize a multi-level clustered dataset. Experimental analyses are described in Sect. 8. We conclude our work in Sect. 9.

## 2 Related Works

Previously, two different types of data structures were proposed for efficiently indexing massive data objects in a vector space. Space partitioning idea was adapted for *grid file* [8], *K-D-B-tree* [9] and *quadtree* [10] that divided the data space along predetermined lines. Data-partitioning index trees such as *R-tree* [11], *R+-tree* [12], *R*-tree* [13], *X-tree* [14], *SR-tree* [15], *M-tree* [16], *TV-tree* [17] and *hB-tree* [18] divided the data space based on the distribution of data objects. These indexing methods worked well for a low-dimensional space, but most of them showed worse performance in higher-dimensional datasets [1,2,19].

*VA-file* [1] was the first indexing scheme that claimed to show better performance than brute-force linear scan in a high-dimensional similarity search. It tried to use compression techniques to improve the query performance. Since it quantized each dimension separately, the resulting regions became rectangular hyperboxes, which resulted in only a suboptimal partition of the data space. *VQ-Index* [3] tried to fix this problem by maintaining complex polytopes which represented data more accurately.

Recently, *LSH* [4,6] has become one of the best-known algorithms for a high-dimensional similarity search. It used a family of locality sensitive hash functions that hashed nearby objects into the same bucket with a high probability. For a given query, the bucket where it was hashed became the candidate set of similar objects. To achieve high search accuracy, *LSH* needed to use multiple (sometimes hundreds of) hash tables to generate a good candidate set.

## 3 Preliminary

In this section, we define two types of nearest neighbor search problems in a formal way, and introduce novel concepts of a search node and its *ring* structure that is used to find the nearest search node.

### 3.1 Problem Settings

We first define several terms that are commonly used in this paper. Euclidean distance is used as a metric in the dataset, but any other metric can be used for the following definitions.

**Definition 1.** *Define $\mathcal{V}$ to be a high-dimensional vector space $\mathbb{R}^d$ where $d \geq 20$ and $\mathcal{D}$ to be a finite set of data points where $\mathcal{D} \subset \mathcal{V}$. Define $dist(p, q)$ to be the Euclidean distance between two points $p$ and $q$ in $\mathcal{V}$. Define $B_p(r)$ to be a ball with radius $r$ centered at $p$ in $\mathcal{V}$.*

We assume the input data is already well-formatted as a high-dimensional vector space. (Developing better ways to extract features from a dataset is out of the scope of this paper.) By doing this, we are able to apply our algorithms to any kind of high-dimensional dataset such as scientific data and multimedia data which can be expressed as a high-dimensional vector space $\mathcal{D}$ defined in Def 1. Note that a query $q$ is a point in $\mathcal{V}$, and a data point is a point in $\mathcal{D}$. That is, it is possible that a query $q$ might be a point which is not a data point.

Now we state our two main problems in this paper. These two problems look similar, but we need to develop different solutions for each problem because of the efficiency issue caused by the large size of the data set. Traditionally, the solution of the $r$-range search problem has been used to solve $k$-nearest neighbor (or $k$-NN) problem, which used bigger $r$ values repeatedly until at least $k$ candidates are obtained and then sorted them to find the $k$ nearest neighbors.

*Problem 1.* (**$r$-Range Search**) Given a radius $r$ and a query $q$, find all data points of $\mathcal{D}$ that reside in $B_q(r)$.

*Problem 2.* (**$k$-NN Search**) Given a constant $k$ and a query $q$, find $k$ nearest data points of $q$ in $\mathcal{D}$.

## 3.2 Search Node

In $\mathcal{V}$, we choose $N$ points, called *search nodes*, that facilitates an efficient nearest neighbor search. Making all data points in $\mathcal{D}$ search nodes will cause a memory issue since each search node entails additional structures explained in Section 3.3. In this paper, we partition the data set $\mathcal{D}$ into $N$ clusters and define *search nodes* as their centers. The radius of a cluster will be used to prune the search space of the nearest neighbors.

**Definition 2.** *Let $C$ be a cluster of data points in $\mathcal{D}$ and $p$ be its center. We say $p$ is a* search node. *Denote $C(p)$ to be $C$ and $\mathcal{S}$ to be a set of all search nodes. Define* radius(C(p)) *as the maximum distance between the center $p$ of the cluster $C(p)$ and the data points in $C(p)$. That is, $radius(C(p)) = \max(\{dist(p, q) | q \in C(p)\})$. Define $size(C(p))$ as the number of data points in a cluster $C(p)$.*

There is a trade-off between the size of the cluster and the performance. Small sized clusters imply a small number of candidates that need to be checked or sorted which leads to faster response time. To make clusters small, we usually need a big number of clusters which requires more computations to conduct clustering algorithms. This burden appears only once as a preprocessing step, so we prefer small sized clusters for better performance. To achieve extreme performance, we apply multi-level clustering for big sized dense clusters since we might have dense clusters even for a large number of clusters. We utilize this hierarchical structure in Section 7 to achieve better performance.

To find the nearest neighbors of a query point, we conduct a search on $\mathcal{S}$ first, and then refine the search to the data points. A search node retains a ring structure for this procedure, and sometimes we use additional data structure, a list of the $m$ nearest search nodes, for a faster search.

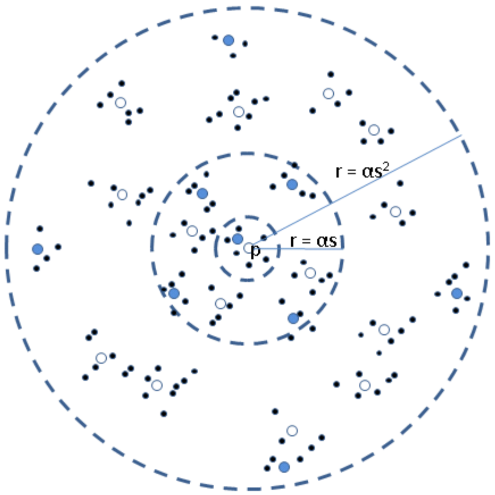


**Fig. 1.** Multi-layered ring structure with exponentially increasing radii

**Definition 3.** *For a search node p, we define* m-NS(p) *as a list of m nearest search nodes of p.*

## 3.3 MLR-Index

We adapt the concept of multi-layered ring structure from [7] to efficiently find the nearest search node (or its cluster) of a given query $q$. Each search node keeps track of a small, fixed number of other search nodes in $\mathcal{V}$ and organizes the list of peers into concentric, non-overlapping rings. This ring structure favors nearby neighbors by providing information on search nodes in the immediate vicinity. Moreover, its exponentially increasing ring radii enable a sufficient number of out-pointers to far away regions to facilitate rapid search.

**Definition 4.** *For a search node p, the i-th ring has inner radius $r_i = \alpha s^{i-1}$ and outer radius $R_i = \alpha s^i$ for $0 < i < i^*$ where $i^*$ is a user defined parameter. For the innermost ring with $i = 0$, we define $r_0 = 0$ and $R_0 = \alpha$. All rings with $i \geq i^*$ are collapsed into a single outermost ring with $r_{i^*} = \alpha s^{i^*}$ and $R_{i^*} = \infty$.*

Each ring defined above keeps track of at most $M$ search nodes in it. If there are more than $M$ search nodes, then the subset of $M$ search nodes that forms the polytope with the largest hypervolume are selected. Nodes that are geographically diverse instead of clustered together enable a node to forward a query to a larger region.

The number of search nodes per ring, $M$, represents the trade-off between performance and overhead. A large $M$ increases the accuracy and the search speed by providing better query routing, but entails more memory at the same time. In [7], it is proved that by the use of a multi-layer ring with $M = O(log(N))$, we can find the nearest search node in $O(log(N))$ steps where $N$ is the number of search nodes in $\mathcal{V}$. Note that $N$ is much smaller than the total number of data points in $\mathcal{D}$.

```
Algorithm FNSN
input: A query point $q$
output: Nearest search node $p$ of $q$
begin
1.  randomly choose any search node $p$
2.  $d \leftarrow dist(p, q)$, $\widetilde{d} \leftarrow \infty$
3.  while $d < \widetilde{d}$
4.      $\widetilde{d} \leftarrow d$
5.      $d \leftarrow$ the minimum distance between the search nodes in $Ring(p)$ and $q$
6.      if $d < \widetilde{d}$
7.          $p \leftarrow$ the search node with the minimum distance d
8.  output $p$
end
```

**Fig. 2.** Finding Nearest Search Node

Finally, we define *MLR-Index* which are composed of all structures mentioned above.

**Definition 5.** *We define* MLR-Index *to be a set of all search nodes together with their ring structures and radius values. For a search node p, we denote* MLR-Index(p) *to be p's ring structure together with* $radius(C(p))$*. Optionally, we add m-NS structure of each search node into* MLR-Index *to improve time efficiency.*

The total memory usage of *MLR-Index* is $N \times ((i^* + 1) \times M + m) = O(Nlog(N))$ for $M = O(log(N))$. Since $N \ll |\mathcal{D}|$, we can claim the space efficiency of *MLR-Index*.

## 4 Finding Nearest Search Node

To answer a range query or $k$-NN search, we begin with finding the nearest search node $p$ of a query point $q$. The *FNSN* algorithm described in Fig.2 shows the procedure of finding the nearest search node. First, we randomly choose a search node $p$ and measure the distance between $p$ and $q$. Then, we compute the minimum distance between $q$ and the search nodes in *MLR-Index* of $p$. If we find a closer search node, then find the minimum distance between its index structure and $q$. This procedure is repeated until we cannot find any closer search node.

We extend the *FNSN* algorithm with an additional input parameter $A$ which retrieves the nearest search node except the nodes in $A$. In this way, we can find the second nearest search node $\widetilde{p}$ of $q$ by applying *FNSN* algorithm twice: $p \leftarrow$ *FNSN*$(q)$ and $\widetilde{p} \leftarrow$ *FNSN*$(q, \{p\})$. Similarly, repeating the procedure described above $m$ times, we can find $m$-th nearest search node of $q$.

A theoretical bound that guarantees logarithmic performance of this procedure is shown in [7]. For $N$ search nodes, the running time of *FNSN* becomes

$O(log(N))$ if each ring retains $M = O(log(N))$ search nodes, and sometimes can assure to find even the exact nearest search node if several conditions are satisfied. In the experiments, we show that a small $M$ is sufficient for a fast response with high accuracy.

## 5 Basic Similarity Search

In this section, we propose two methods *BRS1* and *BKS1* for similarity search mainly using *MLR-Index* and the *FNSN* algorithm. The main idea is to execute *FNSN* on *MLR-Index* repeatedly until there exists no more cluster that intersects with the candidate solution set. Due to the logarithmic performance of *FNSN*, these basic algorithms work efficiently for a small number of iterations. Maximizing the candidate set (in a reasonable way for fast response time) ensures high accuracy performance, but later in Sect.6 we will discuss using an additional data structure that facilitates higher time efficiency by reducing the size of the candidate set. The details of *BRS1* and *BKS1* are shown in the following subsections.

### 5.1 *r*-Range Search

The *BRS1* algorithm described in Fig.3 is the basic algorithm for the $r$-range search problem that uses *MLR-Index*. It sequentially finds next nearest search nodes until there is no more search node whose cluster intersects with $B_q(r)$ for a given query $q$ and a query range $r$. The union of the clusters which intersect with $r$-range search area $B_q(r)$ becomes a set $\widetilde{C}$ whose data points form a candidate set of the range search query. Since each cluster has a small number of data points as mentioned in Sect.3.2, the candidate set is also in a small size which enables efficient computation.

```
Algorithm BRS1
input: A query point $q$ and a range parameter $r$
output: Data points within $B_q(r)$
begin
1.    $p \leftarrow FNSN(q)$
2.    while $dist(p,q) - radius(C(p)) \leq r$
3.        $\widetilde{C} \leftarrow \widetilde{C} \cup C(p)$
4.        $S \leftarrow S \cup \{p\}$
5.        $p \leftarrow FNSN(q, S)$
6.    output points in $\widetilde{C}$ which lies within $B_q(r)$
end
```

**Fig. 3.** Basic $r$-Range Search Using Next Nearest Search Node

The following theorem shows the theoretical legitimacy of this approach.

**Theorem 1.** *Let $q$ be a query and $p_i$ be the $i$-th nearest search node of $q$. Let $R$ be the maximum radius of all clusters. Suppose $j$ is the smallest number such that $dist(q, p_j)$ becomes bigger than $R + r$. Then, there exists no $i$ that is bigger than $j$ such that any data point in $C(p_i)$ lies within $B_q(r)$.*

*Proof.* Proof by contradiction. Suppose that $i > j$ and there exists a data point $a$ in $C(p_i) \cap B_q(r)$. Then, $r \geq dist(q, a) \geq dist(q, p_i) - R \geq dist(q, p_j) - R > r$. This results in a contradiction. Hence, proved.

In the experiments, we found that it was enough to use $radius(C(p))$ instead of the maximum radius $R$ of all clusters. Since usually a small number of search nodes near the query point is sufficient for high accuracy, we might conduct *FNSN* for a fixed number of iterations for a reasonable accuracy with faster response time. Note that, we iteratively find the next nearest search nodes, not retaining them into the memory in advance.

## 5.2 *k*-NN Search

The *BKS1* algorithm described in Fig.4 is the basic algorithm for the $k$-NN search problem that uses *MLR-Index* structure. As we did for range search, this method

```
Algorithm BKS1
input: A query point $q$ and $k$
output: $k$-NN data points of $q$
begin
1.   $p \leftarrow FNSN(q)$
2.   if $|C(p)| \geq k$
3.       $d \leftarrow$ dist(q,$k^{th}$ nearest point of $q$ in C(p))
4.       $\widetilde{C} \leftarrow$ apply BRS1 with $r = d$
5.   else
6.       $A \leftarrow C(p)$
7.       $S \leftarrow S \cup \{p\}$
8.       while $|A| < k$
9.           $\widetilde{p} \leftarrow FNSN(q, S)$
10.          $S \leftarrow S \cup \{\widetilde{p}\}$
11.          $A \leftarrow C(\widetilde{p})$
12.      $d \leftarrow$ dist(q,$k^{th}$ nearest point of $q$ in A)
13.      $\widetilde{C} \leftarrow$ apply BRS1 with $r = d$
14.  output $k$-NN points of $q$ within $\widetilde{C}$
end
```

**Fig. 4.** Basic $k$-NN Search Using Next Nearest Search Node

also executes the *FNSN* algorithm repeatedly to find nearest search nodes of a query $q$. First, we find the nearest search node $p$ by applying *FNSN.* If $C(p)$ contains at least $k$ points, then find the $k^{th}$ nearest point of $q$ within $C(p)$ and draw a ball $B_q(r)$ considering its distance as a radius $r$. Then apply the *BRS1* method with the resulted radius $r$ and find the $k$ nearest data points among them.

If $C(p)$ contains less than $k$ points, then execute the *FNSN* algorithm until the union $A$ of the clusters of the next nearest search nodes of $q$ contains at least $k$ points. Then find the $k^{th}$ nearest point of $q$ within $A$ and draw a ball $B_q(r)$ considering its distance as a radius $r$. Then apply the *BRS1* method with the resulted radius $r$ and find the $k$ nearest data points among them.

The following theorem shows the theoretical legitimacy of this approach.

**Theorem 2.** *Let $q$ be a query and $p_i$ be the $i$-th nearest search node of $q$. Let $R$ be the maximum radius of all clusters. Let $j$ be the smallest number such that $|\bigcup_{i\leq j} C(p_i)|$ contains at least $k$ data points. Let $o$ be the $k^{th}$ nearest data point of $q$ in $|\bigcup_{i\leq j} C(p_i)|$ and $d$ be the distance between $o$ and $q$. Suppose that $m$ is the smallest number such that $dist(q, p_m)$ becomes bigger than $R + d$. Then, there exists no $i$ which is bigger than $m$ such that $C(p_i)$ contains any of the $k$ nearest data points of $q$.*

*Proof.* Proof by contradiction. Suppose $i > m$ and that there exists a data point $\hat{o}$ in $C(p_i)$ which is one of $k$-NN of $q$. Then, $dist(q, \hat{o}) \geq dist(q, p_m) - R > d$. But $B_q(d)$ already contains at least $k$ data points. This results in a contradiction. Hence, proved.

In *BKS1*, we do not have to retain additional structures like $m$-NS that require more memory usage. The running time of *FNSN* is $O(log(N))$ and it becomes more efficient to find nearest search nodes in a series, since they are not far away from each other. Usually, we need to find only a few next nearest search nodes to achieve high accuracy. Note that, similar to *BRS1*, we found that $radius(C(p))$ was enough to use instead of the maximum radius $R$ of all clusters in the experiments.

## 6 *m*-NS Based Similarity Search

In this section, we propose two methods *BRS2* and *BKS2* for similarity search which uses additional structure $m$-NS (***m*-N**earest **S**earch Nodes) for *MLR-Index.* The main idea is to expand the search space by utilizing $m$-NS instead of repeatedly executing *FNSN* on *MLR-Index* to achieve faster response time. Even though *FNSN* shows logarithmic performance, a large number of *FNSN* iterations must be a bottleneck of the performance of *BRS1* and *BKS1.* The details of *BRS2* and *BKS2* are shown in the following subsections.

### 6.1 *r*-Range Search

Based on the heuristic that nearest search nodes of a query point $q$ and those of $q$'s nearest search node $p$ are quite common, we add $m$-NS list of each search

```
Algorithm BRS2
input: A query point $q$ and a range parameter $r$
output: Data points within $B_q(r)$
begin
1.    $p \leftarrow FNSN(q)$
2.    for $\widetilde{p} \in m\text{-}NS$
3.        if $dist(\widetilde{p}, q) - radius(C(\widetilde{p})) < r$
4.            $\widetilde{C} \leftarrow \widetilde{C} \cup C(\widetilde{p})$
5.    output points in $\widetilde{C}$ which lies within $B_q(r)$
end
```

**Fig. 5.** Basic $r$-Range Search Using $m$-NS

node together into *MLR-Index* for a faster search with similar quality of results. The details of this method is described in Fig 5. We first use *FNSN* algorithm to find the nearest search node $p$ of a given query $q$. Then, we find the search nodes among $m$-NS whose clusters intersect with $B_q(r)$. By doing so, we get a set $\widetilde{C}$ of union of those clusters, whose data points form a candidate set of the range query. Finally, the desired points are found within $\widetilde{C}$. In this way, we can reduce the search space from the whole dataset to a small number of clusters. This algorithm *BRS2* uses an additional precomputed data structure $m$-NS for each node to get a faster response with similar accuracy performance to *BRS1*.

## 6.2 $k$-NN Search

As we did for a range search problem, we can also achieve faster response time than *BKS1* by use of an additional data structure $m$-NS of each search node. The main procedures of this algorithm *BKS2* are described in Fig 6. At first, we execute the *FNSN* algorithm to find the nearest search node $p$ from a query point $q$. If $C(p)$ contains at least $k$ points, then find the $k^{th}$ nearest point of $q$ within $C(p)$ and draw a ball $B_q(r)$ considering its distance as a radius $r$. Include the clusters of $m$-NS$(p)$ search nodes which intersect with $B_q(r)$ into the candidate set $\widetilde{C}$.

If $C(p)$ contains less than $k$ points but the union of clusters $A$ of $\widetilde{m}$-NS$(p)$ contains at least $k$ points ($\widetilde{m} \leq m$), then find the $k^{th}$ nearest point of $q$ within $A$ and draw a ball $B_q(r)$ considering its distance as a radius $r$. Include the clusters of $m$-NS$(p)$ search nodes which intersect with $B_q(r)$ into the candidate set $\widetilde{C}$.

Sometimes, we might have a case where even the union of the clusters of $m$-NS$(p)$ contains less than $k$ points. In this case, since we cannot find $k$-NN using the clusters of $m$-NS, we use the *FNSN* algorithm to find more clusters near the query point $q$. Only in this case, we cannot utilize a precomputed $m$-NS list of nearest search nodes. But since this case rarely happens, we still can use this algorithm for the purpose of higher efficiency.

```
Algorithm BKS2
input: A query point $q$ and $k$
output: $k$-NN data points of $q$
begin
1.  $p \leftarrow FNSN(q)$
2.  if $|C(p)| \geq k$
3.      $\widetilde{C} \leftarrow k$-NN of $q$ within C(p)
4.      $d \leftarrow$ dist(q,$k^{th}$ nearest point of $q$ in C(p))
5.      for $\widetilde{p} \in m\text{-}NS(p)$
6.          if $dist(\widetilde{p}, q) - radius(C(\widetilde{p})) < d$
7.              $\widetilde{C} \leftarrow \widetilde{C} \cup C(\widetilde{p})$
8.  else if |clusters of $m\text{-}NS(p)| < k$
9.      apply FNSN repeatedly until we get $\widetilde{m}$ such that |clusters of $\widetilde{m}\text{-}NS(p)| \geq k$
10.     $\widetilde{C} \leftarrow$ union of clusters of $\widetilde{m}\text{-}NS(p)$
11. else
12.     find smallest $\widetilde{m}$ such that |clusters of $\widetilde{m}\text{-}NS(p)| \geq k$
13.     $A \leftarrow$ union of clusters of $\widetilde{m}\text{-}NS(p)$
14.     $\widetilde{C} \leftarrow k$-NN of $q$ within A
15.     $d \leftarrow$ dist(q,$k^{th}$ nearest point of $q$ in A)
16.     for $\widetilde{p} \in m\text{-}NS(p) - \widetilde{m}\text{-}NS(p)$
17.         if $dist(\widetilde{p}, q) - radius(C(\widetilde{p})) < d$
18.             $\widetilde{C} \leftarrow \widetilde{C} \cup C(\widetilde{p})$
19. output $k$-NN points of $q$ within $\widetilde{C}$
end
```

**Fig. 6.** Basic $k$-NN Search Using $m$-NS

## 7 Hierarchical Similarity Search

As previously mentioned, we do the partitioning on the dataset as the vector quantization for efficient data compression. If a cluster is dense, then we may partition it again into smaller clusters to enable more efficient search. In this section, we provide algorithms that utilize this kind of hierarchical property. Note that we extend the *FNSN* algorithm to work in a restricted domain to enable searching the nearest node within a partition.

### 7.1 $r$-Range Search

To answer a hierarchical range query, we apply the basic range query algorithms levelwise. We first find high level clusters that intersect with a query range $B_q(r)$, and for the resulted clusters we perform basic range query algorithms again to retrieve data points within radius $r$ of a query point $q$. In this way, we can perform the search efficiently even though the first level of clusters contain many data points. That is, instead of performing a brute-force linear scan on a

```
Algorithm HRS
input: A query point $q$ and a range parameter $r$
output: Data points within $B_q(r)$
begin
1.    find high level clusters that intersects with $B_q(r)$ using BRS
2.    for each high level cluster $C$ found above
3.        apply BRS on $C$
4.    output data points within $B_q(r)$
end
```

**Fig. 7.** Hierarchical $r$-Range Search

```
Algorithm HKS
input: A query point $q$ and $k$
output: $k$-NN data points of $q$
begin
1.    $p \leftarrow FNSN(q)$
2.    apply BKS within C(p)
3.    $d \leftarrow$ dist(q,$k^{th}$ nearest point of $q$ in C(p))
4.    apply HRS with parameter $d$
5.    output $k$-NN points of $q$
end
```

**Fig. 8.** Hierarchical $k$-NN Search

large number of data points of a first level cluster, we do a logarithmic search on its smaller partitions and perform a linear scan on a reduced set of candidates which minimizes the computational cost.

## 7.2 *k*-NN Search

To search the $k$ nearest neighbors of a given query $q$ in a hierarchical structure, we make use of all algorithms we developed so far. We first find the nearest search node $p$ at the first level. Then we apply *BKS* within the first-level cluster $C(p)$ and get the $k$ nearest points of $q$ from it. We calculate the $k$-th minimum distance $d$ from $q$ to points in $C(p)$, and based on $d$ we apply our hierarchical range search algorithm *HRS* to the entire space except $C(p)$. Finally, we combine the results and output the $k$ nearest data points of $q$ by sorting them in ascending order based on the distance between $q$ and them.

Note that we can apply any version of the *BRS* algorithms in the hierarchical search algorithms *HRS* and *HKS*.

## 8 Experiments

This section presents extensive experimental evaluations on our proposed algorithms. Experiments were conducted on a computer with the following configuration: CPU Intel Pentium 4 2.26GHz, Memory 2GB and OS Redhat Linux 2.6.18 Kernel. Two standard high-dimensional datasets, *Corel* [20] and *CovType* [21], were used for the experiments from *UCI* repository of machine learning databases. The *Corel* dataset contains image features extracted from a Corel image collection. The four sets of features are based on the color histogram, color histogram layout, color moments, and co-occurence. *CovType* was originally proposed to predict forest cover type from cartographic variables which were derived from US Geological Survey (USGS) and US Forest Service (USFS) data. Table 1 summarizes the features of two datasets.

**Table 1.** Statistics for datasets used in experiments

| **Dataset** | **Size** | **Dimensions** | **Application** |
|---|---|---|---|
| Corel | 66,616 | 89 | Content based image retrieval |
| ConType | 581,012 | 54 | Finding similar forest cover type |

To evaluate time performance, we measured the average execution time per query. To evaluate retrieval quality, we used the recall measure. Linear scan was used for the exact solutions of recall measure. We did not show precision and error rate results in this paper because they showed similar tendencies with recall. We used linear scan and the most recent version of *LSH* [6] for the comparisons with our proposed algorithms. *LSH* has a parameter to control the speed and the quality, and we set it as 0.5 in our experiments as higher query speed results in lower retrieval quality.

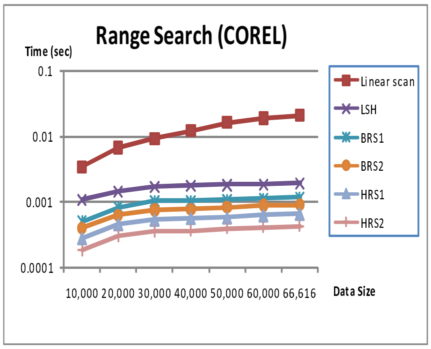
### 8.1 r-Range Search

Fig.9 shows the average query time of $r$-range search on two different datasets *Corel* and *CovType*. Our algorithms show discriminant performance over linear scan and *LSH*. Note that the vertical axes in Fig.9 are in logarithmic scales.

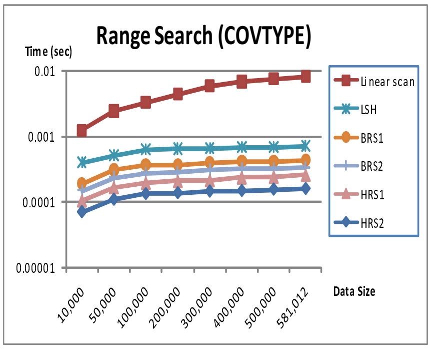
The following experiments used *Corel* dataset with $size = 66616$, $r = 3$, $m = 3$, $i^* = 5$, $M = 20$, and $N = 2000$ by default.

Fig.10 shows the average query times and the corresponding recall values for various $r$ values. For a small query range $r$, most of the algorithms show similar performances, but once $r$ becomes bigger, our proposed algorithms show better performances than linear scan and *LSH*. It is noticeable that *LSH* showed slower query time and lower recall than our algorithms for each $r$ value.

Fig.11(a) and Fig.11(b) show the average query times and the corresponding recall values for various $M$ values. Here, $M$ is the number of search nodes per ring. If the ring retains more search nodes, it must show faster and more accurate results, which turned out to be true.

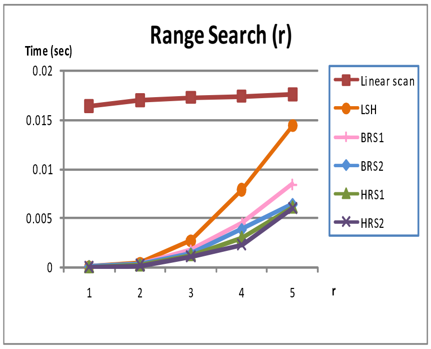


(a) Range search running time for COREL Dataset

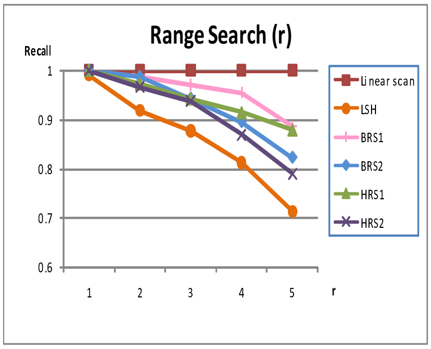


(b) Range search running time for COVTYPE Dataset

**Fig. 9.** Range search running time for various datasets



(a) Range search running time for various r values



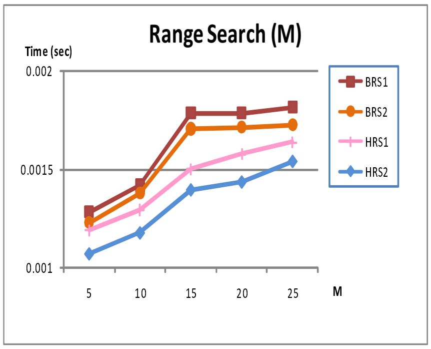
(b) Range search recall for various r values

**Fig. 10.** Range search for various r values

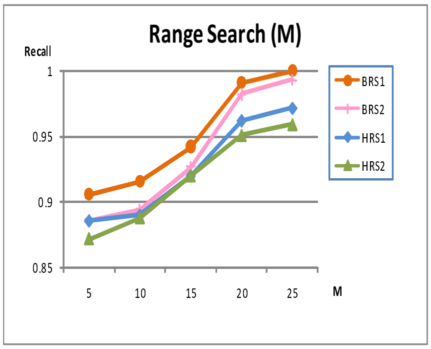
Fig. 11(c) and Fig. 11(d) show the average query times and the corresponding recall values for various $N$ values. Here, $N$ is the number of clusters, or total number of search nodes in a dataset. As the number of clusters becomes bigger, the size of each cluster becomes smaller which leads to higher accuracy but a slower response time because it needs to search more clusters.

For faster response time, we developed $m$-NS structure which is used in *BRS2* and *HRS2*. Fig. 11(e) and Fig. 11(f) show that $m$-NS structure improves not only the response time but also the accuracy of the algorithms. Retaining 2-NS showed higher than 96% of accuracy for both methods.

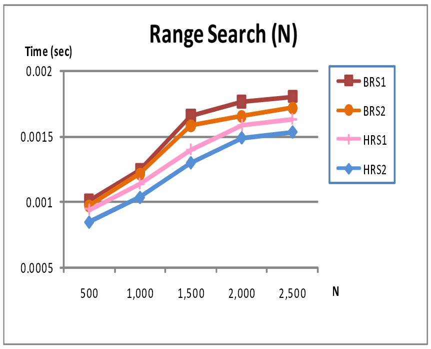
To achieve higher performance, we can increase our two parameters $M$ and $m$, but it also causes a bigger amount of memory usage.



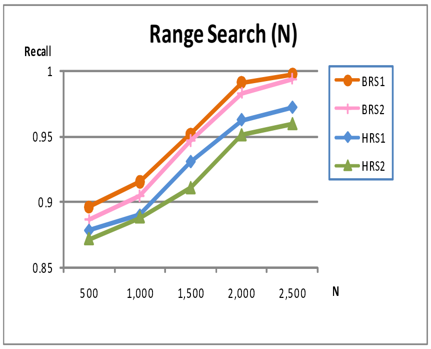
(a) Range search running time for various M values



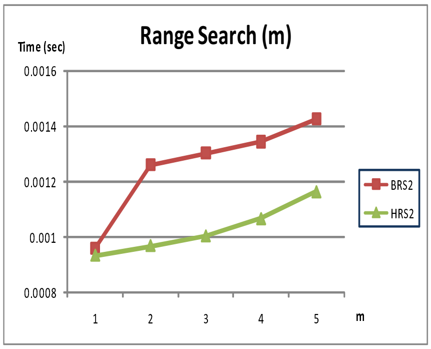
(b) Range search recall for various M values



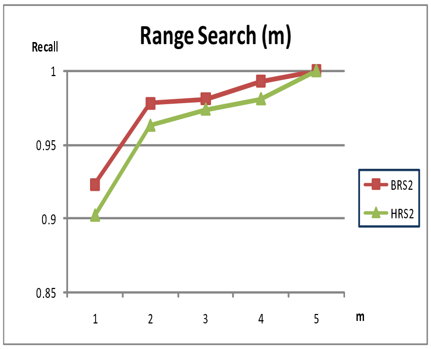
(c) Range search running time for various N values



(d) Range search recall for various N values



(e) Range search running time for various m values

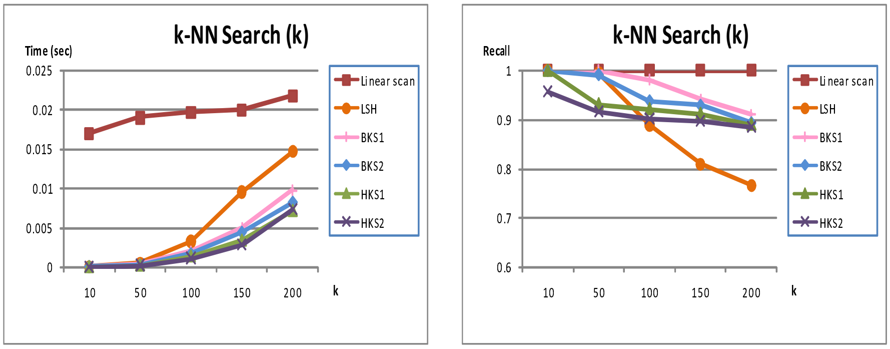


(f) Range search recall for various m values

**Fig. 11.** Range search for various system variables

## 8.2 k-NN Search

The $k$-NN search experiments were done by randomly choosing 100 samples from each dataset. Because of the page limit, we only analyze several results over different $k$ values in this section, since $k$-NN search showed similar tendencies with $r$-range search for various settings of our system parameters.



(a) k-NN search running time for various k values

(b) k-NN search recall for various k values

**Fig. 12.** k-NN search for various k values

Fig. 12 shows the average query times and the corresponding recall values for various $k$ values. For a small number of $k \leq 50$, our proposed algorithms show comparable performances to *LSH*, but once $k$ becomes bigger, our algorithms maintain high accuracy in a stable way while *LSH* dramatically drops down its accuracy.

Overall, we observe the following results for both $r$-range search and $k$-NN search algorithms:

– Using the additional $m$-NS structure is faster than only using the *ring* structure as *MLR-Index* to achieve similar retrieval quality, because directly using $m$-NS avoids additional time cost of performing *FNSN* searches repeatedly.
– Hierarchical search algorithms are faster than non-hierarchical algorithms to achieve similar retrieval quality. The reason is that the hierarchical search algorithms enable faster search on dense clusters.
– Our proposed algorithms are significantly faster than linear scan, and also much faster than best-known method *LSH*, even achieving higher accuracy at the same time. For *LSH*, setting different parameter values might make them faster, but it will cause lower accuracy. Since our algorithms achieve better performance both on speed and quality compared with the current setting of parameters, the experimental results are sufficient to prove the advantage of our algorithms.

## 9 Conclusions

In this paper, we proposed a novel index structure *MLR-Index* (**M**ulti-**L**ayer **R**ing-based **Index**) for high-dimensional indexing problems. Since the vector quantization technique and the ring structure of *MLR-Index* made the search space smaller, we developed an algorithm *FNSN* to find the nearest search node efficiently. By use of *MLR-Index* and *FNSN*, we designed several high-performance solutions for the approximate nearest neighbor search problems including the $r$-range search problem and the $k$-NN search problem. We could design even faster search algorithms with similar high accuracy by adding an additional structure $m$-NS (***m***-**N**earest **S**earch Nodes) into *MLR-Index* and utilizing levelwise clusters for dense partitions. Extensive experimental results comparing with *linear scan* and the best-known method *LSH* showed that our approach was effective and efficient with higher accuracy and faster response time.

## References

1. Weber, R., Schek, H.J., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: VLDB, pp. 194–205 (1998)
2. Beyer, K.S., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is "nearest neighbor" meaningful? In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 217–235. Springer, Heidelberg (1999)
3. Tuncel, E., Ferhatosmanoglu, H., Rose, K.: Vq-index: an index structure for similarity searching in multimedia databases. In: MULTIMEDIA, pp. 543–552 (2002)
4. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: VLDB, pp. 518–529 (1999)
5. Jeong, J., Nang, J.: An efficient bitmap indexing method for similarity search in high dimensional multimedia databases 2, 815–818 (2004)
6. Lv, Q., Josephson, W., Wang, Z., Charikar, M., Li, K.: Multi-probe lsh: efficient indexing for high-dimensional similarity search. In: VLDB, pp. 950–961 (2007)
7. Wong, B., Slivkins, A., Sirer, E.G.: Meridian: a lightweight network location service without virtual coordinates. SIGCOMM Comput. Commun. Rev. 35(4), 85–96 (2005)
8. Nievergelt, J., Hinterberger, H., Sevcik, K.C.: The grid file: An adaptable, symmetric multikey file structure. ACM Trans. Database Syst. 9(1), 38–71 (1984)
9. Robinson, J.T.: The k-d-b-tree: a search structure for large multidimensional dynamic indexes. In: SIGMOD, pp. 10–18 (1981)
10. Finkel, R.A., Bentley, J.L.: Quad trees: A data structure for retrieval on composite keys. Acta Inf. 4, 1–9 (1974)
11. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: SIGMOD, pp. 47–57 (1984)
12. Sellis, T.K., Roussopoulos, N., Faloutsos, C.: The r+-tree: A dynamic index for multi-dimensional objects. In: VLDB, pp. 507–518 (1987)
13. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The r*-tree: an efficient and robust access method for points and rectangles. SIGMOD Rec. 19(2), 322–331 (1990)

14. Berchtold, S., Keim, D.A., Kriegel, H.P.: The x-tree: An index structure for high-dimensional data. In: VLDB, pp. 28–39 (1996)
15. Katayama, N., Satoh, S.: The sr-tree: an index structure for high-dimensional nearest neighbor queries. SIGMOD Rec. 26(2), 369–380 (1997)
16. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: VLDB, pp. 426–435 (1997)
17. Lin, K.I., Jagadish, H.V., Faloutsos, C.: The tv-tree: an index structure for high-dimensional data. The VLDB Journal 3(4), 517–542 (1994)
18. Lomet, D.B., Salzberg, B.: The hb-tree: a multiattribute indexing method with good guaranteed performance. ACM Trans. Database Syst. 15(4), 625–658 (1990)
19. Böhm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Comput. Surv. 33(3), 322–373 (2001)
20. Ortega-Binderberger, M., Porkaew, K., Mehrotra, S.: Corel Image Features Data Set (1999), http://archive.ics.uci.edu/ml/datasets/Corel+Image+Features
21. Blackard, J.A., Dean, D.J., Anderson, C.W.: Covertype Data Set (1998), http://archive.ics.uci.edu/ml/datasets/Covertype

# B-Fabric: An Open Source Life Sciences Data Management System

Can Türker, Fuat Akal, Dieter Joho, and Ralph Schlapbach

Functional Genomics Center Zurich, Winterthurerstr 190, 8057 Zurich, Switzerland
{turker,joho,schlapbach}@fgcz.ethz.ch, akal@acm.org

**Abstract.** The advances in information technology boosted life sciences research towards systems biology which aims at studying complex interactions in biological systems in an integrative way. Steadily improving high-throughput instruments (genome sequencers, mass spectrometers etc.) and analysis software produce large amounts of experimental data. In this paper, we report on experiences with developing and running a life sciences data management system in a productive environment.

## 1 Introduction to B-Fabric

As the complexity of the analytical systems in integrated functional genomics or systems biology research has reached a level where specific, isolated application-oriented data management and analysis have become apparently inefficient, a system for integrated management of experimental data and scientific annotations is needed. With *B-Fabric* [7] we have developed such a data management system at the Functional Genomics Center Zurich (FGCZ). B-Fabric is running in daily business for over two years. In its current configuration, it allows to store and annotate all data produced at FGCZ. For a detailed description of B-Fabric, we refer to [1]. Figure 1 sketches the architecture of B-Fabric. It is composed of distributed, loosely-coupled components based on open source technologies. The *B-Fabric repository* stores experimental (raw and processed) data. The *B-Fabric*

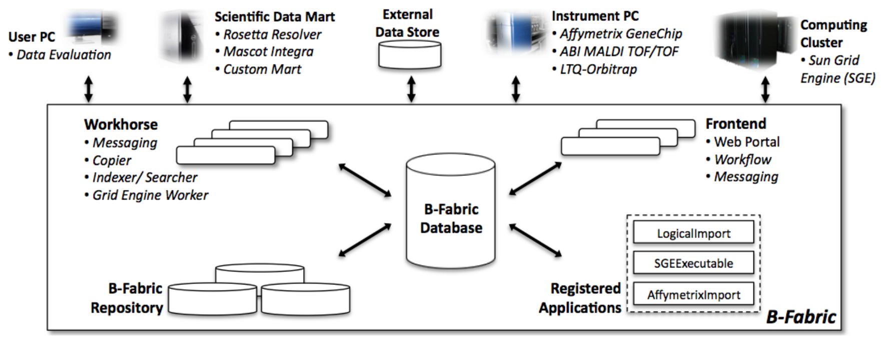


**Fig. 1.** B-Fabric: System Architecture

| | |
|---|---|
| Web Development Framework | Apache Cocoon |
| Database Server | PostgreSQL |
| Object-Relational Mapping | Apache OJB |
| Fulltext Indexing/Search | Apache Lucene |
| Asynchronous Communication | ActiveMQ |
| Workflow Engine | OSWorkflow |
| File Transfer | OpenSSH |
| Logging | Apache log4j |

| | |
|---|---|
| Users | 1207 |
| Institutes | 182 |
| Organizations | 39 |
| Projects | 657 |
| Samples | 2084 |
| Extracts | 2269 |
| Data Resources | 11678 |
| Workunits | 811 |

**Fig. 2.** B-Fabric: Technologies (left) and Data Figures April 2009 (right)

*database* manages all scientific annotations (e.g. about biological samples) and administrative data (e.g. about users and projects). All relevant data is findable via fulltext search. A *frontend* acts as Web portal providing users with controlled access to the data. *Workhorses* execute specific tasks (e.g. data copying, indexing, searching) in a distributed way. Frontends and workhorses communicate via asynchronous messaging. Availability and scalability is achieved by instantiating several frontends and workhorses. The concept of *registered applications* allows an ad-hoc coupling of arbitrary applications with B-Fabric. Using registration profiles, the frontends are dynamically extended with appropriate buttons to invoke and feed the applications with B-Fabric data. These applications may be external ones, autonomously running beyond the control of B-Fabric. Applications can also be invoked within workflows to model scientific pipelines.

B-Fabric interacts with several external components. *User PCs* are standard computers running a Web browser to enable access to B-Fabric through frontends. Typically, a scientist searches and downloads B-Fabric data for analysis reasons. Various data analysis/visualization tools are deployed on these PCs to accomplish those purposes. *Scientific data marts* correspond to external systems that provide scientific functionality. Currently, *Rosetta Resolver* [5] and *Mascot Integra* [3] are used as marts for the detailed management and analysis of transcriptomics and proteomics experiments, respectively. For some instruments, these data marts are not suitable. In such cases, B-Fabric implements a custom data mart to handle this data. *External data stores* represent scientific data available on external systems. At any time, any external data store can be attached to and made accessible via B-Fabric. Users do not need to care about where and how the data is placed. B-Fabric functions as data fabric, capturing and providing the data transparently. *Instrument PCs* refer to computers that are attached to an instrument that generates and holds scientific data to be imported into B-Fabric. To support computationally and data-intensive analysis, B-Fabric can also execute applications on a *computing cluster/grid*.

Figure 2 lists the key technologies glued together in B-Fabric and shows some numbers of the current deployment at FGCZ. Interestingly, 94% of the data resources ($\sim$1.7TB) are stored in external data stores; the remaining data resources ($\sim$2.6GB) are physically imported into the B-Fabric repository. Our users are researchers (usually PhD students interested in running experiments), lab heads (usually professors interested in the experiment results), and FGCZ employees (instrument experts supporting and carrying out the experiments). The latter have special rights, e.g., they are allowed to register applications.

## 2 Experiences with B-Fabric

**Application Integration and Loose Coupling.** Most expert facilities like the FGCZ have applications and workflows for data processing that are now running stable for years. We experienced that moving such functionality into a central system is often a complicated task. First, the original functionality is usually provided by proprietary tools whose internal processing is not known. Second, the scientists and developers that created and installed the application often are no longer available. Third, replacing existing code eats up resources without providing new benefits for the users. These were the reasons why we started implementing connectors to different external applications. However, writing such a connector for every application that had to be connected to B-Fabric was quite laborious. Needs and requests for new applications popped up faster than the implementation could progress. This was the reason why we introduced the *application registration* concept. First, a connector is written for a certain type of external application, e.g. for running *R* scripts on an *Rserve* system [6]. Then, a minimal interface is defined to describe how the external application gets its input. Finally, the scientist writes the external application in any language. The advantage of this approach is that an upgrade or even a replacement of components of an external application is possible without touching the B-Fabric system. Defining the interface towards the external application was quite tricky. Allowing too much flexibility may result in errors since the user usually is not aware of the restrictions of the corresponding external application. Additionally, the heterogenous data that is stored in B-Fabric provides a lot of traps. Figure 3 exemplifies the application integration in B-Fabric. The arrows depict the sequential flow of interaction between a scientist and B-Fabric. The first scenario is an example where the user has high computing needs. `Scientist 1` asks B-Fabric to invoke a previously registered application, called `SGEExecutable` in order to execute his job in the Grid. To meet this demand in our case, B-Fabric accesses the *Sun Grid Engine* running on a compute cluster outside of B-Fabric. The job is executed by the cluster and the results are sent back to B-Fabric.

**Physical and Logical Data Import.** To annotate experimental data, the system must first be aware of the corresponding files. Originally, B-Fabric was designed to move all (relevant) experimental data into its internal repository. Very early in the productive phase of B-Fabric, we could approve that a *physical* import is indispensable for a number of application scenarios, e.g., when the data cannot be maintained persistently at the instrument PC or external data store. However, we also experienced that a physical import into a fully encapsulated data repository has a simple but serious drawback, which was underestimated in the design phase. The use and postprocessing of the data with typical (commercial and prototype) analysis and visualization tools usually require direct access to the data

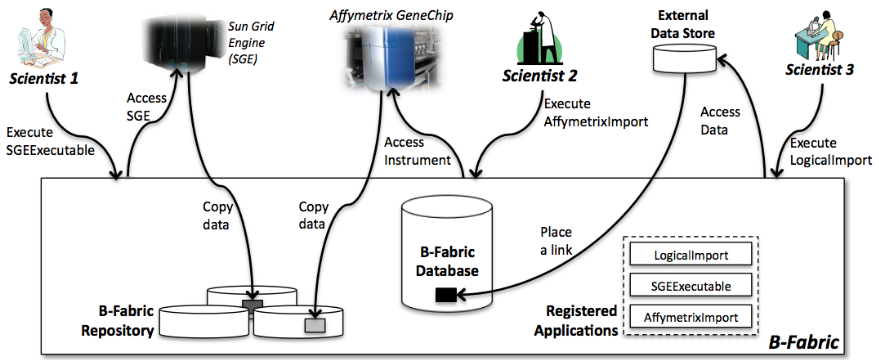


**Fig. 3.** Application and Data Integration

files. Consequently, the physically imported files must be downloaded to the corresponding places in the file system to run the different tools. Due to the size of the files and length of the scientific pipelines, such a file download moves the scientist into an undesired pending state for a while. We learnt our lesson that the files have to be accessible directly by the scientists and tools. Therefore, we adapted B-Fabric to also support a *logical* import (linking) of data from the instrument PCs. Pre-configured data providers take care of the original location of the experimental data on the instrument PC. These data providers transparently support many protocols like *ssh*, *smb*, *jdbc* etc. and hence provide simple and unified data handling. This allows a user to import his data without having any further knowledge about the infrastructure. Figure 3 illustrates both physical and logical data import. `Scientist 2` physically imports his data generated by the *Affymetrix GeneChip* instrument. This is done by invoking a previously registered application, called *AffymetrixImport*. This application accesses the instrument and copies the files into the B-Fabric repository, where it will be under total control of B-Fabric. Note that when coupling an instrument to B-Fabric a corresponding import application has to be registered, too. `Scientist 3` on the other hand logically imports data files from an external data store. For that, the scientist runs the *LogicalImport* application which creates links in B-Fabric to the file at the external data store. Ensuring the consistency, in this case, is a responsibility of the external data store. Since B-Fabric provides a transparent access to the data, scientists are not effected from whether the data is located in the B-Fabric repository or somewhere else. This is achieved by handling files and other data resources through URIs. Each resource knows where its data is located (host and path) and how this data can be accessed (protocol). The fact that 94% of the current data resources maintained in B-Fabric were logically imported shows the importance of this way of managing experimental data.

**External Responsibility.** Although B-Fabric knows the scientific annotations best, it may not be able to interpret and process the experimental data correctly if the instruments produce data in proprietary formats, which is often the case in our experience. This problem is solved in B-Fabric by moving the responsibility of the data processing to the external applications. B-Fabric provides the application input via a small interface. The external application is then able to fetch the information from B-Fabric to process the experimental data correctly. B-Fabric then just waits for the result. This approach requires that B-Fabric trusts the external application to provide the result in a correct way. The external applications need to take additional effort to process the data. Our experiences show that examples and documentation on the development of connectors are vital to support stable and generally accepted applications. However, if a certain level of commitment is achieved, additional functionality can be provided quickly and without changing the B-Fabric interface.

**Data Modeling and Vocabulary.** Many data integration projects failed simply due to the ambitious goal to provide an integrated data scheme that can capture all potentially created data in the application domain at a detailed level of granularity. Already at the beginning of the B-Fabric project, it became clear that in a very dynamic research environment as the FGCZ it is practically impossible to agree on a scheme. Sticking one-to-one with standards such as MIAME [4] or GO [2] was however not a solution. First, there are many researchers arguing that they are doing research on issues that have not been defined yet. Second, depending on the concrete area of the scientists they use different notions and granularities to describe their experimental data. Third, these standards are so huge in size that for most scientist they are impractical for daily use. Therefore, it is vital to have a simple but exact vocabulary. It is a balancing act to use terms that are as precise as needed, but not too specific to end up in an overwhelming set of terms. B-Fabric hence concentrates on a very small data scheme describing only the metadata of central entities such as samples and extracts. After many discussions with our researchers at FGCZ we came to the conclusion that only a small set of commonly agreeable attributes should be fixed for such entities and that the vocabulary defined by the different standards should be initially restricted to that part which is potentially needed for the research technologies supported at FGCZ. To be open to new research directions, B-Fabric allows to dynamically extend the vocabularies at run-time. As soon as a new entry is provided by a user, a reviewing process is started. Depending on the project membership of that user, the coach of that project, in our case an FGCZ employee, is triggered to release or reject the new entry.

**Data Export.** Regardless of the systems functionality, our experience shows that there are always users with needs that go further than the system can support. The best solution to address this issue is to provide an easy access to the data. Scientists should be able to export data and do their calculations on their own. B-Fabric provides several methods for data export. For instance, search results can be exported. Another example is the download of data resources, i.e., files. Due to the large size of the interconnected data, B-Fabric compresses the data and provides a download link for the user. The download of predefined reports is another example. For specific needs and applications, B-Fabric allows reports that collect and prepare data for later use. If it turns out that scientists use a certain type of export very often to perform a specific task, this task becomes a candidate for a B-Fabric internal functionality.

**Data Access Control and Publishing.** Life sciences researchers usually demand for strict data access control while complaining about restrictions in the usage of the data for collaboration purposes. At FGCZ data access is therefore controlled at project level, i.e., project state and membership define access to all the data within a project. While the project has not reached the publish state, the data is visible to the corresponding project members only. This approach works well as soon as the projects are small and have clear and exact goals. Disadvantages of this approach become visible in case of huge projects that run for many years, especially when the scientists involved in the same project should not be able to access each others results, for instance, because they are coming from different facilities. A proven simple but effective solution is to divide such projects into several smaller projects according to their goals and members. In this way, the scientific data can be published as soon as the corresponding research result have been accepted for publication somewhere.

## 3 Outlook

B-Fabric was designed according to the specific needs of the Functional Genomics Center Zurich, for instance, w.r.t. data access policies. To attract the interest of other research groups/centers to use B-Fabric, future releases will generalize it in several directions. First, a more flexible permission management will be implemented. The owner of the data will be allowed to grant and revoke access rights on different data granules even if the corresponding project is not published yet. Second, the data networks will be graphically visualized such that the links and correlations among the scientific data become reproducible.

## References

1. B-Fabric, http://www.bfabric.org/
2. Gene Ontology, http://www.geneontology.org/
3. Mascot Integra, http://www.matrixscience.com/integra.html
4. Minimum Information About a Microarray Experiment - MIAME, http://www.mged.org/Workgroups/MIAME/miame.html
5. Rosetta Resolver System, http://www.rosettabio.com/products/resolver/
6. Rserve, http://www.rforge.net/Rserve/
7. Türker, C., Stolte, E., Joho, D., Schlapbach, R.: B-Fabric: A Data and Application Integration Framework for Life Sciences Research. In: Cohen-Boulakia, S., Tannen, V. (eds.) DILS 2007. LNCS (LNBI), vol. 4544, pp. 37–47. Springer, Heidelberg (2007)

# Design and Implementation of Metadata System in PetaShare

Xinqi Wang and Tevfik Kosar

Department of Computer Science and
Center for Computation & Technology
Louisiana State University
Baton Rouge, LA, 70803, USA
{xinqiwang,kosar}@cct.lsu.edu
http://www.cct.lsu.edu

**Abstract.** As the size of scientific and commercial datasets grows, it becomes imperative that an expressive metadata framework to be developed to facilitate access to the semantics of the data. However, the drive to enhance expressiveness and maximize performance often turns out to be contradictory. Hence, the design of metadata framework needs to take into consideration many factors in order to achieve balance between expressive power and performance. In this paper, we discuss the metadata framework developed for PetaShare and explain the design and implementation decisions we made in the process.

**Keywords:** Metadata, Data Intensive Computing, Ontology, Protege, iRODS, PetaShare.

## 1 Introduction

As the size and complexity of the scientific and commercial datasets continue to grow, new data-centric scientific computing infrastructure needs to be built to satisfy the requirement. Among all the challenges involved in building such computing infrastructure, the need to reconcile the often conflicting requirement for expressive power and sufficient performance is quite challenging.

On the one hand, traditional metadata services have limitations in such application scenarios. Data from different disciplines is difficult to integrate due to lack of expressive metadata framework.

On the other hand, semantically rich representation scheme such as ontology-based semantic metadata provides rich expressiveness while maintaining decidability, but it is not designed for large scale scientific computing. As a result, much of the semantic web [7] technologies developed so far do not provide sufficient performances.

In this paper, we present the metadata system we developed for the PetaShare project. Our system seeks to take advantage of both semantically rich ontology representation of metadata and natively supported metadata represented in triples to provide access for users with different requirements.

We also discuss the choices we made and lessons we learned in the design and implementation process.

## 2 PetaShare

PetaShare [5] is a state-wide distributed data storage and sharing cyberinfrastructure effort in Louisiana. It aims to enable collaborative data-intensive research in different application areas such as coastal and environmental modeling, geospatial analysis, bioinformatics, medical imaging, fluid dynamics, petroleum engineering, numerical relativity, and high energy physics. PetaShare manages the low-level distributed data handling issues, such as data migration, replication, data coherence, and metadata management, so that the domain scientists can focus on their own research and the content of the data rather than how to manage it.

Currently, there are six PetaShare sites online across Louisiana: Louisiana State University, University of New Orleans, University of Louisiana at Lafayette, Tulane University, Louisiana State University-Health Sciencies Center at New Orleans, and Louisiana State University-Shreveport. They are connected to each other via 40Gb/s optical network, called LONI (Louisiana Optical Network Initiative). In total, PetaShare manages more than 250TB of disk storage and 400TB of tape storage across these sites.

At each PetaShare site, there is an iRODS [2] server deployed, which manages the data on that specific site. A Unix-like interface is available to access iRODS storage with commands like “ils”, “icd”, “ichmod” providing functionalities similar to their respective Unix counterparts, “itrim” for trimming number of replicas in iRODS, “irule” for submitting user defined rules and “imeta” for adding, removing, querying user defined metadata. Currently, each iRODS server communicates with a central iCAT [3] server that provides a unified name space across all the PetaShare sites. This centralized system will soon be replaced by a fully distributed and replicated server system. Clients, either human users or programs, can access the PetaShare servers via three different interfaces: petashell, petafs, and pcommands. These interfaces allow the injection of semantic metadata information (i.e. any keywords regarding the content of the file) to the ontology whenever a new file is uploaded to any of the PetaShare sites. The physical metadata information (i.e. file size and location information) is inserted to iCAT using the iRODS API.

As part of the PetaShare project, we intend to develop a semantically-enabled metadata management and query system called petasearch. With petasearch, we intend to design an extendable metadata framework that gives a unified view over multidisciplinary datasets. We also plan to provide fast and efficient metadata query services for physically and conceptually distributed datasets of peta-scale. Protege [1] and iRODS-based metadata management and query system we are going to discuss in this paper are intended to serve as testbed for petasearch. Specifically, we seek to focus on performance issues we encountered during the design and testing of these two systems.

## 3 PetaShare Metadata System

We have implemented two metadata management and query systems in PetaShare, one is based on Protege [1] and and the other one based on iRODS.

As the current de facto standard of ontology design, Protege provides a complete set of tools in support of the design and implementation of ontology-based systems. But the complex nature of the ontologies and the implementation of Protege turn out to be inadequate to provide sufficient performance.

iRODS, on the other hand, is a integral part of PetaShare and it provides its own native metadata system, but the triple-based metadata representation is not powerful enough to satisfy the need for cross-domain metadata management.

Both systems provide some unique advantages the other system currently does not support. At the same time, both systems turned out to be inadequate on other fronts. There are two different approaches we considered to bridge these differences:

1. Take advantage of iRODS's built-in metadata support and try to integrate our semantically rich metadata into iRODS's metadata system. Advantages of this approach include potentially better query performance because of fewer layers a query has to go through. This approach also comes with disadvantages, namely, the added requirement to figure out a way to encode semantic metadata into iRODS's metadata system which, so far, only supports simple queries over triples.
2. Build middleware between current mostly Java-based Semantic Web infrastructure such as Protege where metadata will be modeled and stored and traditionally C-based iRODS where actual data archives are managed. Advantages of this approach includes more established support for ontology insertion, modification, merging and query which our system can readily tap into. Disadvantages may involve developing a whole set of tools required to bridge the inherent differences, also our preliminary testing of the browser based ontology system indicated less than satisfactory performance.

To better understand the advantages and deficiencies of the two approaches, we decided to implement two different metadata systems based on the two different approaches respectively. Our hope is at the current stage, the two systems can complement each other. Eventually, our development goal is to overcome the shortcomings of both systems and hopefully merge the two systems into a unified petasearch metadata system.

### 3.1 Protege-Based Metadata System Framework

The reason we chose to implement the upper layer of our metadata system based on Protege-API and the Protege-based database back-end is to take advantage of the semantic expressive power of the ontologies. As the de facto standard for ontology design, Protege supports almost all the W3C standards and provides support for the whole range of ontology related functionalities, from graphic ontology design interface to built-in reasoner all the way to ontology serialization into relational database, which makes Protege and the Protege-related technologies good candidates for implementing a semantic enabled metadata system.

As shown in Figure 1, two different interfaces are available in our system. They are browser-based and commandline-based respectively. The purpose of browser-based metadata interface to PetaShare is to provide an easy-to-use, easy-to-understand method of access so that scientists can query and obtain small numbers of experimental files, while commandline-based interface can be combined with scripts and other programming tools so that more flexible, more powerful access to bulk files is also available in our system.

The core of the system consists of Protege Query Parser and Semantic Metadata Store. Protege Query Parser is implemented to parse queries entered by users into Sparql [6] queries understandable to the Protege query engine. In Semantic Metadata Store, metadata definitions in the forms of ontological classes and ontological instances are stored. Protege itself provides two ways of storing ontologies: file-based and relational database-based. The first approach essentially stores ontological classes and instance definitions to text files, although it is easier to implement and access file-based ontology, our experiment showed that file-based ontology can not scale to satisfy the data intensive requirements

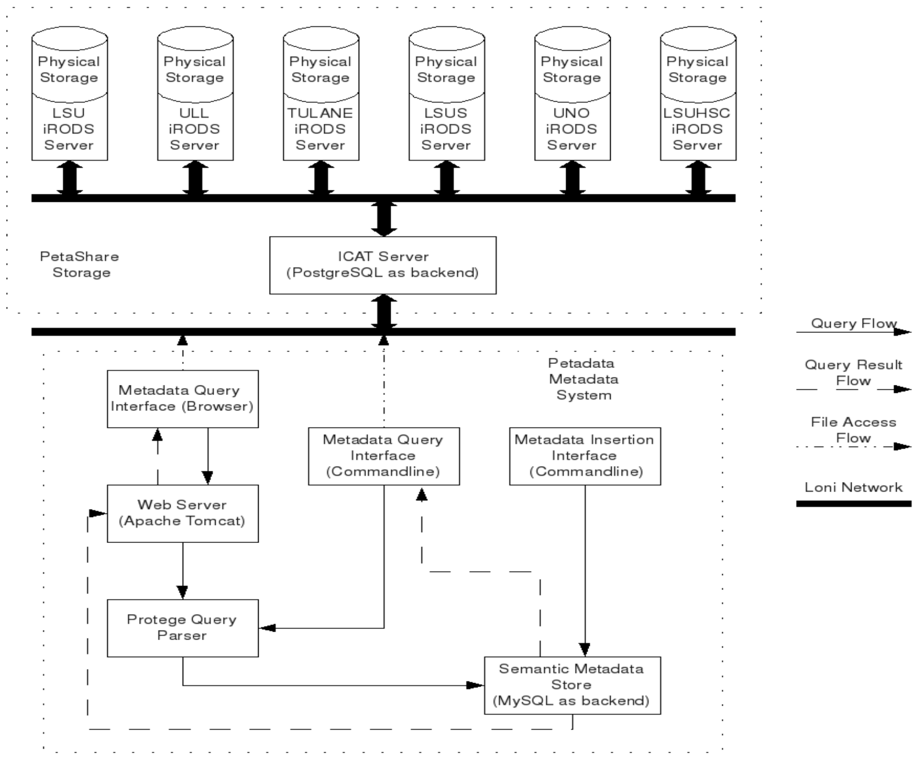


**Fig. 1.** PetaShare Metadata Framework for Protege

of PetaShare: attempts to insert metadata instances in excess of ten thousand resulted in insufficient memory error. Even though increase of physical memory size can partially alleviate this problem, the fact that the Java Virtual Machine places limit on the amount of physical memory it can handle means text-based ontology can not scale as much as we want. Another problem is it often takes more than a dozen hours to load text-based ontology with more than ten thousand instances into memory. The causes of the failure to scale include:

1. The amount of memory required exceeds the maximum memory size the Java Virtual Machine is capable of handling.
2. System is saddled with too high a performance overhead as a result of large numbers of file accesses.

To overcome the above mentioned problems, we decided to take advantage of the second approach and store our ontology in regular relational databases. In our system, we chose MySQL as the back end database in which all metadata are stored in ontological form. We tested insertion of 1 million instances on a workstation with 4 GB memory. Each insertion consisted of creating 15 properties for a particular file in the archive. Our experiment showed that 1 million instances could be inserted in 6898 minutes 59 seconds, approximately 5 days. The relatively slow speed of the insertion owes partially to the overheads associated with representing complex modeling scheme such as ontology through tables in relational database. More experiments are needed to assess the upward limit of relational database-based ontology store.

Another part of our system is called Metadata-insertion interface. It is a Java-based commandline program that can be utilized, with the help of script languages such as perl, to automatically insert metadata about newly created experiment files.

For example, in large science experiments, when an experiment file is created, metadata-insertion interface can be triggered to automatically add appropriate metadata information, such as name, keyword, time of creation, file type, etc, into Semantic Metadata Store. The system administrator can also choose to do bulk-insertion. As of now, we have successfully inserted metadata about more than 1 million files.

Our Protege-based metadata system implementation offers support for ontology-based metadata query, ontology-based automatic metadata insertion, as well as ontology-based file access through both browser and command-line interfaces.

One typical use scenario is as follows: a meteorologist needs some monitoring data on Hurricane Katrina's path of movement. He also would like to see a visualization of the monitoring data. In real life, raw monitoring and visualized data could belong to different projects, and different projects may have different vocabulary for describing data. The use of the ontologies in the Protege-based system can bridge the semantic differences that may exist among different science projects. We assume here that raw and visualized data belong to different projects. In this use scenario, on PetaShare, the meteorologist could simply open his web browser or the specific PetaShare commandline interface. Here we assume he types "Katrina" into the search box of his web browser and presses the search button. The straight arrow in Figure 1 shows the data flow of his query. PetaShare will then search its metadata store and return a list of files from both projects it thinks are related to Hurricane Katrina, as indicated in Figure 1 by dotted arrows. Then the meteorologist can simply click whatever file he wants to obtain, the metadata system will send out request to other parts of PetaShare to fetch the file back into the machine of the meteorologist.

The biggest advantage for the Protege-based system is the establishment of a unified view of scientific data across different science projects or even different science disciplines. A unified data view can enable scientists to access data from multiple projects from multiple disciplines, regardless of the differences in vocabulary. Such data view is critical in modern, increasingly cross-disciplinary science.

The shortcomings of the Protege-based metadata system are clearly illustrated in Figure 1. Basically, in exchange for the expressive power of the ontologies, we have to build another metadata system independent of the iCAT [3] metadata system used by iRODS. Doubtlessly, the extra set of metadata and everything related to its management add overhead to overall performance of PetaShare. Also almost the entire set of technologies we employed to implement the system is Java-based, which introduces more overhead to performance and more complications to achieve maximum scalability.

### 3.2 iRODS-Based Metadata System Framework

Unlike ontology-based system, the iRODS-based Metadata System does not support a richly representative scheme, namely ontology, like Protege does. On the other hand, iRODS and its corresponding iCAT metadata system serve as the backbone of PetaShare. As a result, metadata system based on iRODS and iCAT is naturally integrated into PetaShare seamlessly. Also, unlike ontology technology which is Java-based and was originally designed for Semantic Web with little prior consideration for performance, iRODS and its corresponding metadata system iCAT were designed with the requirements of data-intensive computing in mind. Better performance can be achieved as a result.

As Figure 2 shows, the framework of the iRODS-based metadata system is far simpler. Only one extra layer of system is added to the existing iRODS-based PetaShare storage. The PetaShare clients have been developed to parse and remote-execute various iRODS commands. One such command is "imeta", which is used for inserting and accessing metadata stored in iCAT.

Command "imeta" can be used to insert metadata about iRODS files, collections, resources and users in the form of Attribute-Value-Unit triples (AVUs) Because iCAT also employs relational database as back end storage and the fact that iCAT deals with metadata far less expressive than ontology does, we expect it to be able to be at least as scalable as the Protege-based system. Our experiment indicates that iCAT can easily handle file metadata in the order of millions of files. In current implementation, command "imeta" can only insert metadata one AVU at a time. To expand its functionalities, we implemented another

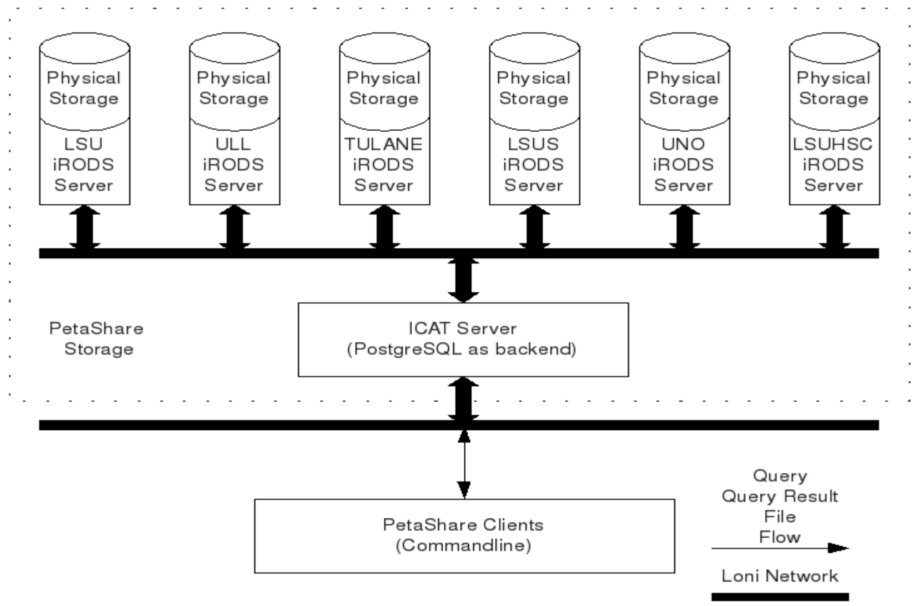


**Fig. 2.** PetaShare Metadata Framework for iRODS

version of command "imeta" that supports bulk-insertion function similar to the one provided by Metadata-insertion interface in the Protege-based system.

In the iRODS-based metadata system, a typical query operation would be users typing in what they want to query as parameters of command "imeta". "imeta" will do the query and return a list of files. Users then can use other iRODS commands supported by the PetaShare clients to access the files needed to be accessed.

There are several functionalities missing in command "imeta". For example, although Attribute-Value-Unit triples are easy to handle and understand, their expressive power is far from being able to adequately meet the requirement of an extendable semantic metadata system we envision for PetaShare. On the other hand, at the lowest level of representation, an ontology also consists of triples. We think it is possible to implement certain middleware so that upper level ontology-based metadata can be stored in lower level iCAT store. We are also interested in adding over the long term, the kind of querying and reasoning capabilities supported by the Protege-based system.

## 4 Issues and Observations in Performance Evaluation Experiments

In this section, we will present our investigation into some of the performance and scalability issues we encountered during the design and implementation process. Both systems have been deployed on PetaShare, with controlled access granted to four PetaShare science projects. The iRODS-based metadata system is used internally by the PetaShare system itself, and the Protege-based system is deployed as a web-based search engine that allows scientists to search the metadata store and access files stored in PetaShare.The scalability and performance of the our deployed systems is not entirely satisfactory, as a result, we conducted an series of experiments to test it.

We first tested the bulk insertion program we developed for the Protege-based system and modified "imeta" command for the iRODS-based system as users often need to first "build up" their presence on PetaShare, which makes performance and scalability of the bulk insertion program a important issue.

In the case of the Protege-based system, the Java implementation of Protege and the complexity of the ontologies exact a heavy toll on performance of the Protege-based insertion. During our experiment, we attempted to test the limit of the scalability of the Protege-based system. The conflict of a Java-based system and the memory requirement for data intensive applications was laid bare: The Java Virtual Machine can only use at most 2 GB of memory in Unix like system, which is hardly enough for a ontology containing metadata for millions of files. We experimented with inserting metadata for 1 million files in the Protege-based system. The experiment ran 19 hours 12 minutes and 46 seconds. It succeeded in inserting metadata for 684632 files, then the process crashed when another Java-based program was launched. Another attempt ended with metadata for 734845 files inserted in 24 hours 43 minutes and 53 seconds. The process crashed again presumably because of memory hog. It has also been observed that as the size of metadata grew, the execution of the insertion programs became extremely slow and unresponsive. Eventually, we succeeded in inserting one million instances after several failed attempts, the insertion took close to 5 days to finish because of various bottlenecks discussed above. Further investigation is needed to determine the exact cause of the performance differences we witnessed and how big the ontology can be scaled to.

It is clear that as the size of the ontologies grows, the Protege-based system would encounter scalability problem. In the future implementation, we plan to physically distribute the ontology to expand its scalability.

Scalability test on the iRODS-based system, however, did not go as smoothly as we hoped either. Similarly to what we did on the Protege-based system, we attempted to insert metadata for 1 million files to the iRODS-based system. We observed that there was not discernible slow-down of the insertion speed as the size of metadata inserted grew, but error occurred after 2 hours 2 minutes and 35 seconds and metadata for 109428 files inserted. The error forced the insertion operation to stop and restart. Restarted insertion operation continued metadata inserting without a glitch until 1 hour 57 minutes and 50 seconds later and another approximately 100000 files inserted when another similar error occurred. Our observation was the iRODS-based system had no problem handling metadata for tens of thousands of files, but the insertion needs to be done in batches with approximately 100000 files as one batch. Precisely what caused the limitation on the size of the metadata insertion operation that can finish in one single run is not clear yet. We will tryto answer this question with more experiments in our follow-up work.

Beside scalability, we also tested performance on querying metadata store on the two systems. Our experiments showed that query time on the iRODS-based system is positively correlated to the size of result set. As the size of the result set grew, significantly more time was needed for the query to finish.

In the case of the Protege-based system, however, performance varied little as the query result size changed. Another observations of ours was that in the Protege-based system, queries that would return tens of thousands of files crashed the system. When no crash occurred, the performance was extremely bad, which indicated that the size of available memory that can be utilized by the Java Virtual Machine is also a contributing factor to query performance of the Protege-based system.

## 5 Conclusion

In this paper, we have introduced two preliminary experimental metadata management systems developed for the PetaShare project. For each system, we have presented the design and underlying technologies, and discussed potential benefits and pitfalls they might bring to the capabilities and performance of the overall metadata system. We have also discussed performance and scalability issues we encountered in the development process.

**Acknowledgement.** This project is in part sponsored by the National Science Foundation under award numbers CNS-0846052 (CAREER), CNS-0619843 (PetaShare) and EPS-0701491 (CyberTools); by DOE under award number DE-FG02-04ER46136 (UCoMS); and by the Board of Regents, State of Louisiana, under Contract Numbers DOE/LEQSF (2004-07), NSF/LEQSF (2007-10)-CyberRII-01, and LEQSF(2007-12)-ENH-PKSFI-PRS-03.

## References

1. Protege, `http://protege.stanford.edu/`
2. iRODS, `https://www.irods.org/`
3. iCAT, `https://www.irods.org/index.php/iCAT`
4. Kosar, T., Livny, M.: Stork: Making Data Placement a First Class Citizen in the Grid. In: ICDCS 2004: Proceedings of the 24th International Conference on Distributed Computing, pp. 342–349. IEEE Computer Society, Washington (2004)
5. Balman, M., Suslu, I.: Distributed data management with PetaShare. In: MG 2008: Proceedings of the 15th ACM Mardi Gras conference, p. 1. ACM, New York (2008)
6. SPARQL Query Language for RDF, `http://www.w3.org/TR/rdf-sparql-query/`
7. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. In: Scientific America (1993)

# Covariant Evolutionary Event Analysis for Base Interaction Prediction Using a Relational Database Management System for RNA

Weijia Xu[1], Stuart Ozer[2], and Robin R. Gutell[3]

[1] Texas Advanced Computing Center,
The University of Texas at Austin, Austin, Texas, USA
xwj@tacc.utexas.edu
[2] One Microsoft Way Redmond, WA., Seattle, Washington, USA
stuarto@microsoft.com
[3] Center of Computational Biology and Bioinformatics
The University of Texas at Austin, Austin, Texas, USA
Robin.gutell@icmb.utexas.edu

**Abstract.** With an increasingly large amount of sequences properly aligned, comparative sequence analysis can accurately identify not only common structures formed by standard base pairing but also new types of structural elements and constraints. However, traditional methods are too computationally expensive to perform well on large scale alignment and less effective with the sequences from diversified phylogenetic classifications. We propose a new approach that utilizes coevolutional rates among pairs of nucleotide positions using phylogenetic and evolutionary relationships of the organisms of aligned sequences. With a novel data schema to manage relevant information within a relational database, our method, implemented with a Microsoft SQL Server 2005, showed 90% sensitivity in identifying base pair interactions among 16S ribosomal RNA sequences from Bacteria, at a scale 40 times bigger and 50% better sensitivity than a previous study. The results also indicated covariation signals for a few sets of cross-strand base stacking pairs in secondary structure helices, and other subtle constraints in the RNA structure.

**Keywords:** Biological database, Bioinformatics, Sequence Analysis, RNA.

## 1 Introduction

Comparative sequence analysis has been successfully utilized to identify RNA structures that are common to different families of properly aligned RNA sequences. Here we present enhance the capabilities of relational database management for comparative sequence analysis through extended data schema and integrative analysis routines. The novel data schema establishes the foundation that analyzes multiple dimensions of RNA sequence, sequence alignment, different aspects of 2D and 3D RNA structure information and phylogenetic/evolution information. The integrative analysis routines are unique, scale to large volumes of data, and provide better accuracy and performance. With these database enhancements, we details the computational approach to score the number of mutual or concurrent changes at two positions in a base pair during the evolution of Bacterial 16S rRNA molecules.

Since early studies on covariation analysis [1-4] analyzed the nucleotide and base pair frequencies, it has been appreciated that measuring the number of evolutionary events increases the accuracy and sensitivity of covariation analysis. Examples include prediction of a pseudoknot helix in the 16S rRNA [4] and determining divergent and convergent evolution of the tetraloop [5]. More commonly, evolutionary events were found anecdotally, suggesting that many more can be identified from a more computationally intensive search for *evolutionary events*.

This particular problem requires the traversal of the phylogenetic tree hierarchy, potentially within only selected phylogenetic branches, joined with the existing multiple sequence alignment at large scale to identify sequence changes at paired positions. Over the years, the number of RNA sequences dramatically increased [6]. The number of sequences in ribosomal RNA alignments have grown from under 12, 000 sequences 5 years ago to nearly 200, 000 sequences today and the growth is accelerating[7]. The traditional flat file storage format for multiple sequence alignment [8] and the phylogenetic information from heterogonous data provenance not only requires custom data accessing methods but also pushes the limits of memory scale and affordability for available commodity hardware .

Our approach enhances the capabilities of a relational database management system (RDBMS), in this case Microsoft SQL Server 2005 [9] to provide a scalable solution for a class of problems in bioinformatics through data schema and integrated analysis routines. The value of integrating biological sequence analysis into relational databases has been highlighted by commercial offerings from Oracle [10] and IBM (DB2) [11]. A number of research projects have made broad fundamental attacks on this problem. Patel et al. have proposed a system extending existing relational database systems to support queries for the retrieval of sequences based on motifs [12]. Miranker et al. intended to support built-in database support for homology search [13-15]. Periscope/SQ system extends SQL to include data types to represent and manipulate local-alignments and together with operators that support both weighted matching of subsequences and, regular expression matching [16].

A key contribution of our work is a data schema that unifies various types and dimensions of RNA information within a single relational database system. Data from various sources, including sequences and their annotations, sequence alignments, different types of higher-order structure associated with each position for each sequence and taxonomy for each of the rRNA sequences are maintained in our RNA Comparative Analysis Database (rCAD)[17]. The data schema of developed in rCAD closely ties together phylogenetic, sequence and structural information into a form that is readily analyzable using relatively simple SQL expressions -- dramatically simplifying a wide family of tasks that used to require complex custom programming against 'closed' data structures. Analysis of these multiple dimensions of data in rCAD is performed within this SQL-server system as stored procedures written in the C# language and T-SQL. With this integrated data storage and analysis system, significant reductions in the runtime were achieved by only analyzing those pairs of positions that might possibly have a significant concurrent change or covariation. Our approach is evaluated for both performance and accuracy of the result and compared with a previous study[18].

## 2 Background and Related Work of Evolutionary Covariation Analysis

### 2.1 Background for Covariation Analysis on Aligned Sequences

Covariation analysis, a specific form of comparative analysis, identifies positions with similar patterns of variation indicating that these sets of positions in a sequence evolve (ie. mutate) at similar times during their evolution, as determined from their phylogenetic relationships. Thus the positional variation is revealing a dependency in the patterns of variation for certain sets of positions in a multiple sequence alignment. And when this dependency is strong we have interpreted that the two positions that covary are base paired with one another. The accuracy of this method was rigorously tested and substantiated from a comparison of the covariation structure model and the high-resolution crystal structures for several different RNA molecules[19].

While the high accuracy of the predicted structure with covariation analysis and the elucidation of tertiary or tertiary-like base-base interactions substantiated the utility of the covariation analysis, previous analyses [2] has also revealed that this form for comparative analysis on a set of aligned sequences can reveal lower degrees of dependency between the patterns of variation for sets of positions that either indicate a base pair, a base triple, or a more complex structural unit that involves more than two positions, for example, hairpin loops with four nucleotides, tetraloops[5].

### 2.2 Related Work

Although the concept and importance of an evolutionary event has been stated in numerous publications [4, 20-28], there are few sophisticated computational methods that can systematically identify them[18, 29]. Both of these methods are recent. Yeang et. al. derive a probabilistic graphical model to detect the coevolution from a given multiple sequence alignment[18]. This approach extends the continuous-time Markov model for substitution at single position to that for substitutions at paired positions. In addition to the computationally expensive model-based computation steps, this approach requires the computation of evolutionary distance among sequences in preprocessing steps using an external program with a computational cost of $O(n^3)$[30, 31]. Therefore, this approach doesn't scale over large multiple sequence alignment and has limited applicability. The experiments reported in [18] are limited to only 146 16S ribosomal RNA sequences sampled from different phylogenetic branches within the Bacteria domain. Dutheil et al. used a Markov model approach to map substitution of a set of aligned sequences to the underlying phylogentic tree. The approach applied to a bacterial rRNA sequences from 79 species[29].

In contrast to these model-based approaches, our method utilizes known phylogenetic relationships and identifies the minimal number of mutual changes for two positions during the evolution of the RNA under study. The major computational objective is to identify the pair of positions with similar patterns of variation and to increase the score for those covariations that have occurred more frequently during the evolution of the RNA sequences. Our method functions on any given multiple sequence alignment and existing phylogenetic classifications without additional preprocessing. With a novel data schema for storing multiple sequence alignments and phylogenetic tree reconstructions in rCAD, our approach exploits the efficiency of the database system and the effectiveness of C# running within the query engine to perform recursive computations on tree structures, thus enabling analysis of more than 4200 Bacterial 16S ribosomal RNA sequences that are distributed across the entire Bacterial phylogenetic domain (approximately 2337 different phylogenetic branches). To the best of our knowledge, this is the largest number of sequences analyzed with a phylogenetic-based covariation method.

## 3 Methods and Implementations

Central to our approach is a novel data schema for managing multiple dimensions of data using a relational database system and a coarse filter to facilitate the online computations. Due to the size of input data, two main challenges are the accessibility of the data and computational feasibility. Effectively accessing a large amount of various types of data is not a trivial step. In this case, each position of a multiple sequence alignment must be easily accessed by either row or column and any branch of the phylogenetic tree must be able to be hierarchically accessible. For computational efficiency and ease of development, our approach integrates the data analysis process directly into the database management system.

Our computational approach includes several steps as illustrated in Figure 1. For a given multiple sequence alignment, a candidate set of pairs of positions (columns) is first selected from all possible pairs. Since it is computationally expensive to compute mutual information for every possible pair over a large alignment, we developed a coarse filter based on our empirical observations. The coarse filter effectively reduces the workload for computing mutual information. Filtered pairs are then evaluated for mutual information content, and candidate pairings are identified from the mutual information scores. Pairs are then predicted by gathering the number of mutation events in each candidate pairs' positions based on the phylogenetic tree. Pairs with potential interactions are then predicated based on the percentage of covariant events of total observed events and the number of total events observed.

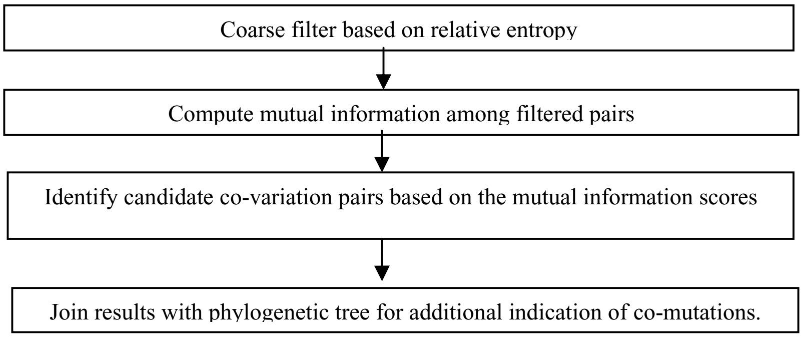
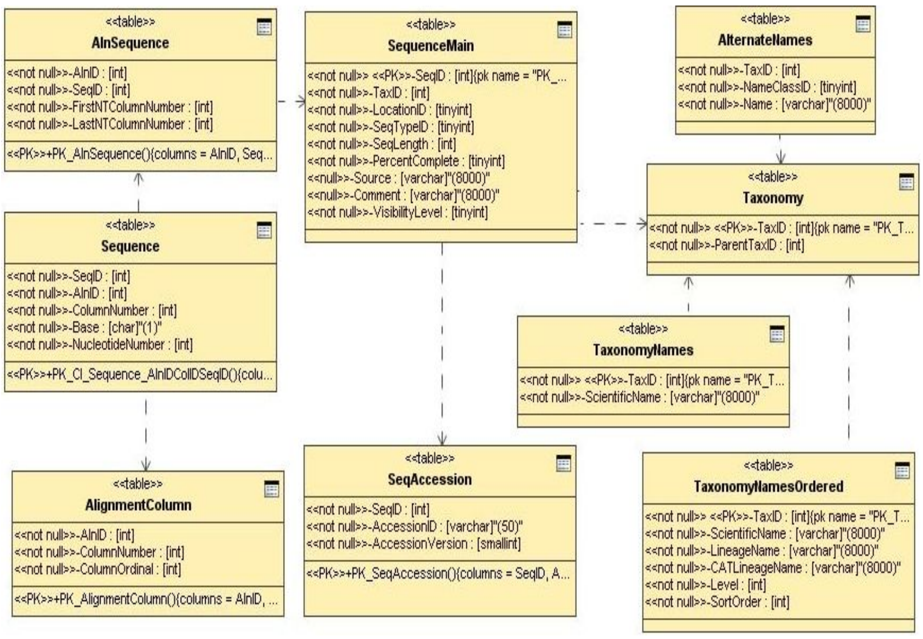

Coarse filter based on relative entropy
Compute mutual information among filtered pairs
Identify candidate co-variation pairs based on the mutual information scores
Join results with phylogenetic tree for additional indication of co-mutations.

**Fig. 1.** Overview of evolutionary event analysis

Any set of sequences can be selected from the existing alignment table based on various criteria with no need to repopulate the database or tables. Additionally, intermediate computing results are stored within the database as tables. Those tables can be used to trace back/repeat an analysis or for repurposing previous computations beyond the initial experimental design. For example, computations on a subset of sequences from a specific taxonomy group maybe derived from results of previous runs with larger scope

## 3.1 Data Schema and Database Design

The design of rCAD is fundamentally based on the idea that no data in the tables need ever be re-stated or re-organized. All access to the alignment data is typically driven by a sub-query that selects the sequence IDs of interest and places those values in a table variable or temp table as part of the analysis. Analysis queries then join these IDs to the sequence data itself present in the SEQUENCE table. Selection of sequences for display, analysis or export is typically specified using the sequences' location in the taxonomy hierarchy, combined with the ID of the type of molecule and alignment of interest. Other properties present in the Sequence Main table or related attributes can also be used for selection. The data structures are designed to hold not only a single alignment, but an entire library of aligned sequences spanning many RNA molecules of many types from many species. Figure 2 shows only tables used to store Sequence metadata, taxonomy data and alignment data that are essential for event counting studies.



<<table>>
AlnSequence
<<not null>>-AlnID : [int]
<<not null>>-SeqID : [int]
<<not null>>-FirstNTColumnNumber : [int]
<<not null>>-LastNTColumnNumber : [int]
<<PK>>+PK_AlnSequence(){columns = AlnID, Seq...
<<table>>
SequenceMain
<<not null>> <<PK>>-SeqID : [int]{pk name = "PK_...
<<not null>>-TaxID : [int]
<<not null>>-LocationID : [tinyint]
<<not null>>-SeqTypeID : [tinyint]
<<not null>>-SeqLength : [int]
<<not null>>-PercentComplete : [tinyint]
<<null>>-Source : [varchar]"(8000)"
<<null>>-Comment : [varchar]"(8000)"
<<not null>>-VisibilityLevel : [tinyint]
<<table>>
AlternateNames
<<not null>>-TaxID : [int]
<<not null>>-NameClassID : [tinyint]
<<not null>>-Name : [varchar]"(8000)"
<<table>>
Taxonomy
<<not null>> <<PK>>-TaxID : [int]{pk name = "PK_T...
<<not null>>-ParentTaxID : [int]
<<table>>
Sequence
<<not null>>-SeqID : [int]
<<not null>>-AlnID : [int]
<<not null>>-ColumnNumber : [int]
<<not null>>-Base : [char]"(1)"
<<not null>>-NucleotideNumber : [int]
<<PK>>+PK_Cl_Sequence_AlnIDColIDSeqID(){colu...
<<table>>
TaxonomyNames
<<not null>> <<PK>>-TaxID : [int]{pk name = "PK_T...
<<not null>>-ScientificName : [varchar]"(8000)"
<<table>>
AlignmentColumn
<<not null>>-AlnID : [int]
<<not null>>-ColumnNumber : [int]
<<not null>>-ColumnOrdinal : [int]
<<PK>>+PK_AlignmentColumn(){columns = AlnID, ...
<<table>>
SeqAccession
<<not null>>-SeqID : [int]
<<not null>>-AccessionID : [varchar]"(50)"
<<not null>>-AccessionVersion : [smallint]
<<PK>>+PK_SeqAccession(){columns = SeqID, A...
<<table>>
TaxonomyNamesOrdered
<<not null>> <<PK>>-TaxID : [int]{pk name = "PK_T...
<<not null>>-ScientificName : [varchar]"(8000)"
<<not null>>-LineageName : [varchar]"(8000)"
<<not null>>-CATLineageName : [varchar]"(8000)"
<<not null>>-Level : [int]
<<not null>>-SortOrder : [int]

**Fig. 2.** Database tables related with event counting

*Sequence Metadata. Information about available RNA sequences is centered on the "SequenceMain"* table, which links each sequence to its Genbank accession numbers, cell location (e.g. mitochondria, chloroplast or nucleus), NCBI Taxonomy assignment, raw (unaligned) sequence, and other source annotations. Each sequence is uniquely identified by a locally-assigned integer *SeqID*.

*Taxonomy Data.* While the NCBI taxonomy assignment of each sequence is present in "*SequenceMain*" via a locally maintained integer *TaxID*, we need two tables "*TaxonomyNames*" and "*AlternateNames*" to associate each *TaxID* with the scientific and alternative names of the taxa. However, these two tables alone cannot provide any evolutionary relations among taxons. The "*Taxonomy*" table retains the basic parent-child relationships defining the phylogenetic tree. When joined with the "*SequenceMain*" table, SQL's ability for handling recursive queries makes it straightforward to perform selections of sequences that lie only on specific branches of the phylogeny tree. We also materialize a derived table, "*TaxonomyNamesOrdered*", that contains, for each taxon, a full description of its ancestral lineage.

*Sequence Alignment.* The bulk of the database records are contained in the tables representing the sequence alignments. Each alignment, identified by an integer *AlnID*, can be thought of conceptually as a 2-dimensional grid, with columns representing structurally aligned positions among sequences, and rows representing the distinct sequences from different species and cell locations. Each family of molecules, such as 16S rRNA, 23S rRNA, or tRNAs correspond to one or possibly more alignments. Our main design concern is to (1) avoid storing data for alignment gaps and (2) minimize the impact of inserting a sequence that introduces new columns within an existing alignment.

The table "*Sequence*" contains data only for positions in each sequence that are populated with a nucleotide. Each entry is keyed by *SeqID* identifying the sequence, *AlnID* identifying the alignment. The field *Base* specifies the nucleotide present at that position of the sequence. The field *NucleotideNumber* is populated with the absolute position of that location within the sequence. The field *ColumnNumber* is a unique integer key assigned to every column of an alignment. With this schema, the *ColumnNumber* associated with an entry in "*Sequence*" never changes even when additional columns are inserted into the alignment. The table "*AlignmentColumn*" associates a *ColumnNumber* and its actual position in an alignment, which is noted as field *ColumnOrdinal*. Thus, if adding a new sequence requires inserting new columns into an alignment, only the values of *ColumnOrdinal* for a small number of entries in "*AlignmentColumn*" need to be updated to reflect the new ordering of columns. No data in the potentially huge and heavily indexed "*Sequence*" table need be changed. Gaps in the alignment are easily identified by the absence of data in the "*Sequence*" table. The table "*AlnSequence*" defines the first and last column number within an alignment for which sequence data exists for a given sequence.

Table indexes are created to facilitate data access. The queries that perform the Relative Entropy calculations and Mutual Information scoring of all candidate column pairs rely on the Clustered Index (alignment ID, Sequence ID, Column ID) to execute a high-performance sequential read covering exactly the range of sequence data of interest. Queries that perform the event counting of candidate column pairs use a covering secondary index (alignment ID, Column ID, Sequence ID) to quickly extract only the data from column pairs of interest within a specific alignment, avoiding unnecessary reads.

### 3.2 An Entropy Based Coarse Filter

The mutual information score is a measure of the reduction of randomness of one position given the knowledge of the other position. A high mutual information score indicates a strong correlation between two positions. The Mutual information score and its variations have been used in several studies [29, 32, 33]. Given a multiple sequence alignment, let x and y denote the two positions of interest, then mutual information (MIXY) between column x and column y is calculated as

$$MI(x;y) = \sum_{i,j} p_{12}(x_i, y_j) \ln \frac{p_{12}(x_i, y_j)}{p_1(x_i) p_2(y_j)}$$

where p(x, y) is the joint probability function of x and y, and p(x) and p(y) is the marginal probability for position x and y.

With a multiple sequence alignment of m columns and n sequences, the time complexity for computing mutual information of each pair of columns is O(n). The total possible column pairs to be considered is in the order of O($m^2$). Hence the total computational cost to computing mutual information among every possible pair is in the order of O($m^2$n). The multiple sequence alignment for 16S bacteria RNA sequence used here consists of 4,000 sequences and 3,237 columns. To speed up this computation, we developed a fast coarse filtering method based on empirical observations. Note that the compositional distribution between covariance columns always shows minimal difference. The compositional distribution difference between two positions x and y can be measured as

$$H(vx \parallel vy) = \sum_{i} p(vx_i) \log_2 \frac{p(vx_i)}{p(vy_i)}$$

Our approach uses modified relative entropy as a coarse filter. Only pairs with relative entropy score less than a predefined threshold are selected for mutual information computation. For each pair of positions, the computation of relative entropy is linear to the number of sequences. As shown in the result section, the relative entropy filter can significantly reduce the search space for mutual information computation and speed up the total running time. However, the probability distribution is easily skewed if the sample size is relative small. This is also the case that we observed here. A number of columns in the alignment contain a large portion of gap symbols rather than nucleotide bases. If there are very few bases in a column, the estimated results become unreliable. Therefore, we also prune out columns in which the number of bases are less than a predefined threshold.

With the filtered set of candidate column pairs, we then perform a mutual information calculation to compute the MIXY score. Candidate base pairs are identified by considering all column pairings (x,y) where the MIXY score for (x,y) is among the top N MIXY scores for all pairs (x,k), and the MIXY score for (x,y) is also among the top N MIXY scores for all pairs (k,y). This process is known as finding the Mutual N-Best MIXY scores, and can be executed as a single SQL query over the set of all scores for candidate pairs. N is typically set as 100~200 to identify a broad range of candidates for phylogenetic event counting analysis.

Sequences used in the input multiple sequence alignments are grouped according to their taxonomy classification. For each candidate pair, we accumulate the number of evolutionary events within each taxonomy group. There are two types of evolutionary events.

**Definition 1 Evolutionary event** Given a pair of positions of a set of aligned sequences and their ancestral sequence, an evolutionary event is observed when there is at least one mutation from its ancestral sequence for each sequence.

**Definition 2 Covariant evolutionary event** Given a pair of positions of a sequence, a covariant evolutionary event is observed when both positions contain mutations from its ancestral sequence for every sequence.

In practice, there are usually no actual ancestral sequences at every internal node of a phylogeny tree. Therefore, given an arbitrary set of sequences, we define an equality set for each position as candidates for ancestral sequences.

**Definition 3 Equality set** Given a multiple sequence alignment, an equality set is defined as the set of base(s) with the maximum number of occurrences at those column(s) in the multiple sequence alignment.

```
1. Start with root
2. If a node is a leaf
3.  Compute the equality set for the node as the set of pair
      types having the maximum count among the child sequences
      at this node
4. else
5.  Compute the equality set for the node as the set of pair
      types having the maximum of (count of pairs of this
      type among child sequences at this node + count of
      pairs of this type among child equality sets).
6. For each member of equality set as candidate parent
7.     For each child sequence.
8.         If the child is different from the candidate parent,
9.              Count an evolutionary event.
10         If the child covaries with the candidate parent,
11              Count a covariation event.
12     For each child equality set.
13       If every member of the set differs from the parent,
14              Count an evolutionary event.
15       If every member covaries with the candidate parent
16              Count a covariation event.
17 EventCount := minimum event count among the tested parents.
18 CovaryCount := minimum test covariation count among the
                  parents.
```

**Fig. 3.** Pseudo code of evolutionary event counting algorithm

Figure 3 shows pseudo code of the event counting algorithm. We use a variation of Fitch's maximum parsimony approach adapted for non-binary trees since any node in the phylogeny tree can contain an arbitrary number of sequences. We first determine the type of pair represented by a parent node, as the *equality set,* which may be a mixture of pair types (Lines 2-5). Based on the *equality set* of a node, we then compute the number of events and covariations exhibited by children at that node

(Lines 7-11). We only count a pair type once per occurrence in a child equality set and do not count the number of underlying descendant pairs of that that type (Lines 12-16). The event count resulting from choosing any member of the equality set as the parent will be the same as the count from any other parent. However, since covariant counts will differ based on parent selections; we only considered the minimum count in this case (Lines 17,18).

The algorithm is implemented as a C# stored procedure that bridges the 'impedance mismatch' between the types of queries that the relational model natively supports, and computations desired on more complex data structures such as graphs and trees. The Event Counting procedure initially takes the parent-child relationships expressed in the Taxonomy table and populates a high-performance recursive set of .Net Treenode data structures at query execution time -- which is then used to easily compute the event counts. This is an important illustration of how we can extend the expressiveness of SQL Queries to embody a rich computation close to the data.

## 4 Performance and Experiments Results

The size of alignment considered here is significantly larger than other related works and poses a major computational challenge. The alignment data used in this paper are publicly available at the CRW website[34]. The alignment is generated and curated by human experts over the past 20 years. New sequences have been periodically added into this alignment through a progressive alignment approach. The Bacteria 16S rRNA alignment consists of 4200 sequences, 3237 columns, and 6,103,090 bases. The phylogeny tree is downloaded from NCBI and stored into relational tables as described in the section 2. The phylogeny tree data is also updated periodically. At the time of this experiment, the sequences were from 2337 different classification groups.

### 4.1 Database Performance Evaluations

Figure 4 shows an overview of the execution time of the entire computational process. The entire computing process took over 28 hours. The evolutionary event counting step accounts for 98.9% computing time. The covariation computation and filtering steps account for 77.90% and 19.77% of the rest of the computations. Based on the computational complexity, the entire computation consists of two parts. The first part is to create a list of candidate pairs based on an existing multiple sequence alignment. The second part is to calculate the evolutionary events for the candidate pairs. The computational complexity of the first part depends on the number of sequences and the length of the aligned sequences. The computational complexity of the second part depends on the size of the candidate pairs and number of taxons in the taxonomy tree. In practice, the second part of the computation accounts for 99% of the computations.

Our approach enables a flexible way to select multiple sequence alignments from existing data. Using the SQL selection query, an arbitrary 'window' view of an alignment can be created from a subset of alignments or a range of columns. The selection query can be further combined with information stored in the other tables to construct a customized data set such as one based on composition statistics or phylogenetic information. With these features, we created a series of subsets of alignments to study the database performance.

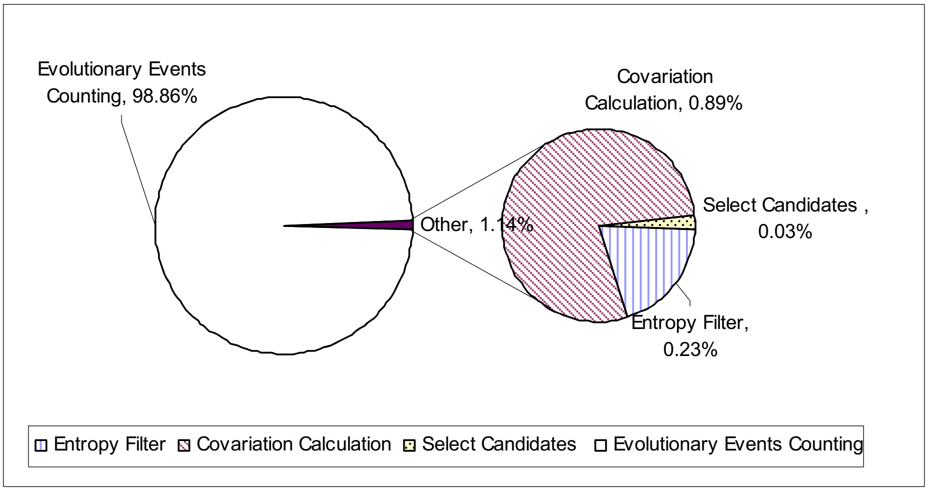


**Fig. 4.** Execution time breaks down

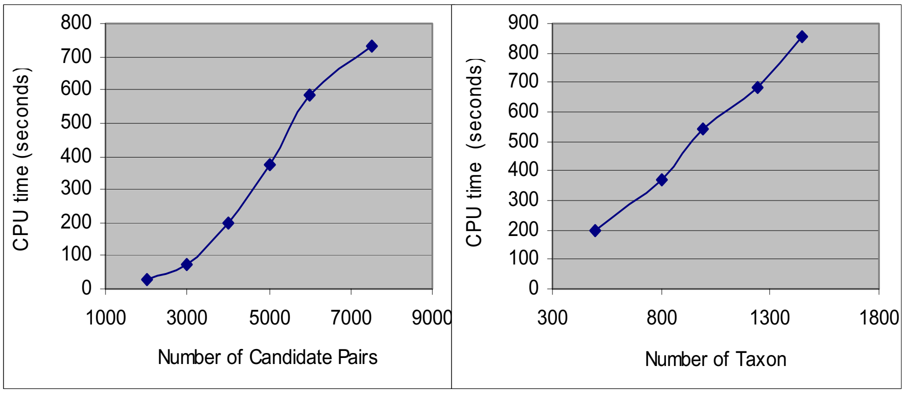


**Fig. 5.** Evolutionary counting performances vs. size of candidate pairs (left) and number of Taxons (right) considered in computations

Figure 5 (left) shows the evolutionary counting part of its performance with regard to the number of candidate pairs. In this test, we selected a subset of sequences which are covered by a taxonomy tree with 496 taxons. We then only select a fixed number of pairs from the set of candidate pairs. The CPU time is reported through a feature of the MS SQL server. Figure 5 (right) shows the evolutionary counting performance with regard to the number of taxons considered. For each testing subset of sequences, the number of taxons required various from 496 to 1450. The number of candidate pairs used in the computations is limited at 4000 for each test set.

### 4.2 Effectiveness of Coarse Filtering Methods.

To speed up covariation computation, we implemented a coarse filter to select pair candidates. In the coarse filtering step, each column in a multiple sequence alignment is inspected with two criteria: the number of bases contained within each column and the relative entropy of that column.

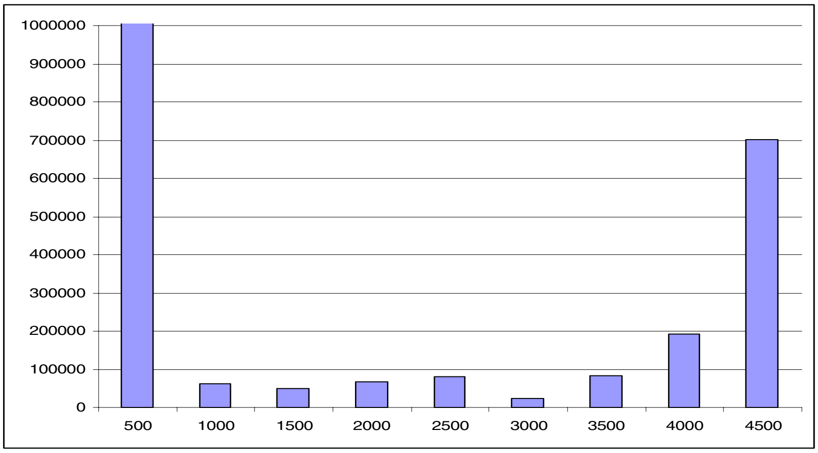


**Fig. 6.** Number of possible pairs at each position grouped by the minimum number of bases

A multiple sequence alignment can be viewed as a matrix with a fixed number of columns. Each sequence, as a row in this matrix, is filled with gaps in order to proper ly align conserved bases. As more diversified sequences across the phylogeny tree are aligned, the number of gaps is expected to increase. From a practical consideration, a position with a large number of gaps, i.e. a small number of bases, is a unlikely to be found in a covariant pair.

Figure 6 shows the number of possible pairs grouped by the minimum number of bases contained in each position for each five hundred interval. For example, the group labeled as 2500 shows that there are 79,255 possible pairs in which the minimum number of bases of the two positions is between 2000 and 2500. Note that the column for the first 500-group is truncated for presentation. There are 3,924,099 pairs which contain positions with less than 500 non-gap bases, and this accounts for about 75% of the total 5,182,590 possible pairs. For each pair in the first 500-group, each pair contains at least one column filled with more than an 88% gap. From practical experience, we pruned those pairs for further consideration.

The second score used in coarse filtering is the relative entropy score described in the previous section. For each column, the relative entropy is computed. Pairs with a maximum relative entropy value higher than a given threshold are pruned.

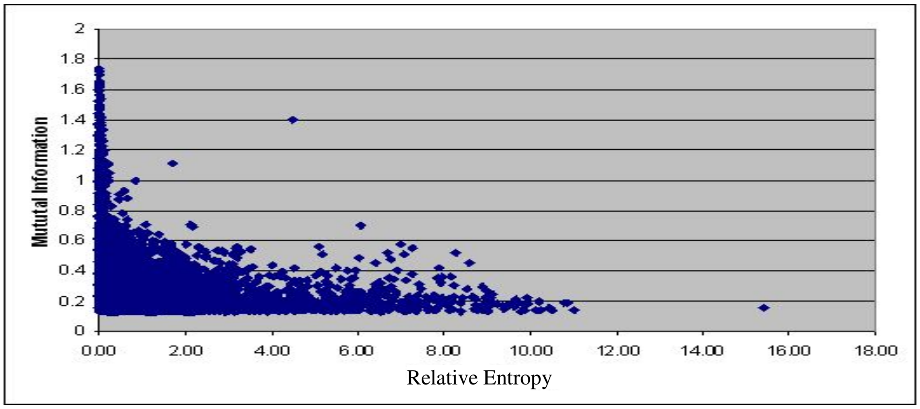


**Fig. 7.** Relative entropy score vs. mutual information score

**Table 1.** The number of pairs selected by the filters for Bacteria 16S rRNA dataset

| | |
|---|---|
| Total possible pairs | 5,234,230 |
| Pairs selected with more than 500 minimum number of base occurrences | 1,258,491 |
| Pairs selected based on combined base occurrences and relative entropy less than 0.5 | 587,747 |

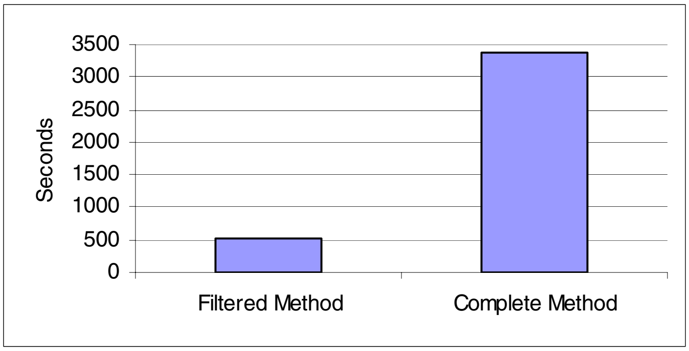


**Fig. 8.** Comparison of running time for filtered method and complete method

Figure 7 plots 51,820 pairs from all possible pairwise comparisons for the Bacteria 16S rRNA alignment. The mutual information for each selected pair is higher than 0.1. The plot shows a good correspondence between maximum relative entropy score and mutual information scores with only a handful of outliers. From practical experience, we determined a threshold of 0.5 for relative entropy score computations (Table 1). Figure 8 shows the comparison of total running time for covariant analysis with and without coarse filters.

## 4.3 Performance Comparison with CO Model

Since our approach is fully integrated within the relational database system and the scale of data considered here, a precise performance comparison with related work which is implemented in functional programming language is not straightforward. Yeang et al. implemented the model computation in C and reported over 5 hours of running time for 146 sequences[18]. However, this running time does not include the computation of evolutionary distance among sequences using DNAPARS in the PHYLIP package[35]. In our test using the same code and data set provided by the authors, the same computation takes 6 hours and 1 minute to run in command prompt using our sql server machine. However, for the 4200 16S rRNA sequences used in our study, both preprocessing and model computation are failed to finish within the 10 days time frame. Since our approach was finished within 28-hour time frames, we conclude our method is more efficient than related work due to the fact the other method failed at running on the scale of our dataset.

## 4.4 Experimental Results Evaluations

From the annotations maintained at CRW, we expect to find 357 secondary base pairing interactions and 17 tertiary pairing interactions in our predicated data set. In our approach, a pair of positions is considered as coevolved if there are more than 50% of co-evolutionary events among all accumulated evolutionary events. Figure 9 shows the percentage of annotated base pairs when using the percentage of covariation events. The figure shows very high selectivity. For example, there are 244 out of 271 (or 90%) of pairs with more than 50% of covariation events are known base pairs. Therefore, our approach can achieve 90% accuracy in identifying known interacting pairs. By comparison, Yeang et al. reported the 57% sensitivity of using a proposed

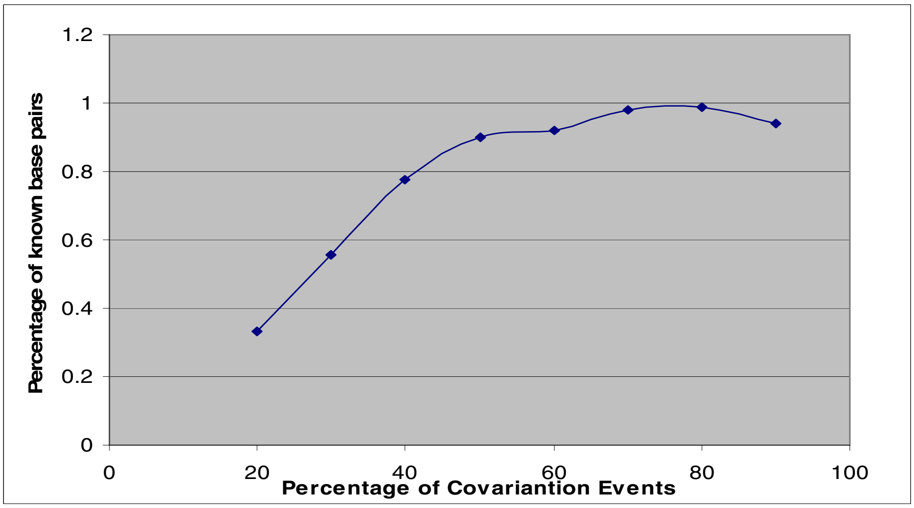


**Fig. 9.** Accuracy of using percentage of covariation events for indentify nucleotide interaction

CO model to identify 251 secondary interactions with 150 false positives and much lower sensitivities with other models including the Watson–Crick co-evolution model and using mutual information.

In addition to the results that are consistent to pairs with known base pairing interactions, our results also indicate candidate pairs that have lower constraints (or non-independence) on the evolution of RNA sequences. While the vast majority of the covariations have been interpreted correctly as a base pair interaction between the two positions with similar patterns of variation[19], exemplar interactions between non-base paired nucleotides have been identified in the literature such as base triple tertiary interactions in helix and tetraloops [5, 36]. In those situations, a number of nucleotides within the same helix or loop also show a tendency of coevolution over time but are not base paired with one another. Instead of forming hydrogen bonding, the bases of these nucleotides are partially stacked onto one another or same strand or cross strand. However, those interacting nucleotide sites are often filtered out by traditional covariation analysis due to a lack of contrast from the background.

## 5 Conclusions and Discussions

While the significant increase in the number of nucleic acid sequences creates the opportunity to decipher very subtle patterns in biology, this overwhelming amount of information is creating new challenges in the management and analysis with computational methods. Our approach enhances the capability of a modern relational database management system for comparative sequence analysis in bioinformatics through extended data schema and integrated analysis routines. Consequently, the features of RDBMS also improve the solutions of a set of analysis tasks. For example with dynamic memory resources management, alignment sizes are no longer limited by the amount of memory on a server. And analysis routines no longer need to incur the runtime overhead of loading entire alignments into RAM before calculations can begin. As a result alignments can now be analyzed in the order of thousands of columns and sequences to support comparative analysis.

The extended data schema presented here is not designed specifically for the covariant evolutionary event analysis. It is designed to facilitate a class of sequence analysis tasks based on multiple sequence alignment such as computing template multiple sequence alignments, and performing conventional covariation analysis. It is the data schema like this that enables us to conduct innovative analysis on tasks such as covariant evolutionary event counting presented in this paper. Several other innovative analysis methods enabled by this data schema are being developed by our research group, including statistical analysis on sequence compositions and structures. However, we won't be able to present all of those related research tasks in complete detail in this paper.

Furthermore, integrating analysis within the relational database environment has enabled simpler, faster, more flexible and scalable analytics than were possible before. Now concise SQL queries can substitute for pages of custom C++ or script programming. To store analytical results in relational tables simplifies the process of forming an analysis pipeline. New analysis opportunities also emerge when results from different query, parameters can be joined together or with updated data. We expect the database management system to play a key role in automating comparative sequence analysis.

While the computational improvements noted above make it possible to analyze significantly larger datasets and integrate different dimensions of information, our analysis reveals that this newer comparative analysis system has increased sensitivity and accuracy. Our preliminary results are identifying base pairs with higher accuracy, and with higher sensitivity we are finding that a larger set of nucleotides are coordinated in their evolution. Based on our previous analysis of base triples and tetraloop hairpin loops[5, 36], these larger sets of coordinated base changes are forming more complex and dynamic structures. This approach also has the potential to identify different base pairs and other structural constraints in the same region of the RNAs in organisms on different branches of the phylogenetic tree. Previously, a few examples of this from have been identified through anecdotal visual analysis of RNA sequence alignments. For example positions 1357:1365 in the *Escherchia coli* 16S rRNA co-vary between A:G and G:A in bacteria, and G:C and A:U pairings in Archaea and Eucarya [37, 38]. In conclusion, with significant increases in the number of available RNA sequences and these new computational methods, we are confident in our ability to increase the accuracy of base pair prediction and identify new structural elements.

Central to both biological analysis and computational algorithm development is the issue of how various types of information can be used most effectively. The increasing complexity of analyzing biological data is not only caused by the exponential growth of raw data but also due to the demand of analyzing diverse types of information for a particular problem. For *in silico* explorations, this usually means manually marshalling large volumes of related information and developing a new application program for each problem or even problem instance (or at least the *ad hoc* scripting and custom parameterization and integration of existing tools). The size of the data also makes everything, from retrieving relevant information to analyzing large result sets, no longer trivial tasks. Thus biologists will benefit from an integrated data management and analysis environment that can simplify the process of biological discovery. As the growth of biological data is outpacing that of computer power, disk-based solutions are inevitable (i.e. the Moore's law doubling constant for biological data is smaller than the constant for computer chips) [6, 39]. With mature disk-based data access machineries and a well designed data schema, a relational database management system (DBMS) can be an effective tool for bioinformatics.

**Acknowledgements.** This work is support with grants from the NIH (GM085337 and GM067317) and from Microsoft Research. The database servers are maintained by Texas Advanced Computing Center, a TeraGrid resource provider.

## References

1. Cannone, J.J., et al.: The comparative RNA web (CRW) site: an online database of comparative sequence and structure information for ribosomal, intron, and other RNAs. BMC Bioinformatics 3(1), 2 (2002)
2. Gutell, R.R., et al.: Identifying constraints on the higher-order structure of RNA: continued development and application of comparative sequence analysis methods. Nucleic Acids Res. 20(21), 5785–5795 (1992)

3. Gutell, R.R., et al.: Comparative anatomy of 16-S-like ribosomal RNA. Prog. Nucleic Acid Res. Mol. Biol. 32, 155–216 (1985)
4. Gutell, R.R., Noller, H.F., Woese, C.R.: Higher order structure in ribosomal RNA. EMBO J. 5(5), 1111–1113 (1986)
5. Woese, C.R., Winker, S., Gutell, R.R.: Architecture of ribosomal RNA: constraints on the sequence of tetra-loops. Proceedings of the National Academy of Sciences of the United States of America 87(21), 8467–8471 (1990)
6. Benson, D.A., et al.: GenBank. Nucleic acids research, 35(Database issue), pp. D21–D25 (2007)
7. Walker, E.: A Distributed File System for a Wide-area High Performance Computing Infrastructure. In: Proceedings of the 3rd conference on USENIX Workshop on Real, Large Distributed Systems, vol. 3. USENIX Association, Seattle (2006)
8. Macke, T.J.: The Ae2 Alignment Editor (1992)
9. Walker, E.: Creating Private Network Overlays for High Performance Scientific Computing. In: ACM/IFIP/USENIX International Middleware Conference. ACM, Newport Beach (2007)
10. Stephens, S.M., et al.: Oracle Database 10g: a platform for BLAST search and Regular Expression pattern matching in life sciences. Nucl. Acids Res. 33, 675–679 (2005)
11. Eckman, B.: Efficient Access to BLAST using IBM DB2 Information Integrator. IBM health care & life sciences (September 2004)
12. Tata, S., Patel, J.M.: PiQA: an algebra for querying protein data sets. In: Proceedings of 15th International Conference on Scientific and Statistical Database Management, pp. 141–150 (2003)
13. Miranker, D.P., Xu, W., Mao, R.: MoBIoS: a Metric-Space DBMS to Support Biological Discovery. In: 15th International Conference on Scientific and Statistical Database Management (SSDBM 2003), Cambridge, Massachusetts, USA. IEEE Computer Society, Los Alamitos (2003)
14. Xu, W., et al.: Using MoBIoS' Scalable Genome Joins to Find Conserved Primer Pair Candidates Between Two Genomes. Bioinformatics 20, i371–i378 (2004)
15. Xu, W., et al.: On integrating peptide sequence analysis and relational distance-based indexing. In: IEEE 6th Symposium on Bioinformatics and Bioengineering (BIBE 2006), Arlington, VA, USA (accepted) (2006)
16. Sandeep, T., James, S.F., Anand, S.: Declarative Querying for Biological Sequences. In: Proceedings of the 22nd International Conference on Data Engineering (ICDE 2006). IEEE Computer Society, Los Alamitos (2006)
17. rCAD: Comparative RNA Analysis Database (manuscripts in preparation)
18. Yeang, C.H., et al.: Detecting the coevolution of biosequences–an example of RNA interaction prediction. Molecular biology and evolution 24(9), 2119–2131 (2007)
19. Gutell, R.R., Lee, J.C., Cannone, J.J.: The accuracy of ribosomal RNA comparative structure models. Curr. Opin. Struct. Biol. 12(3), 301–310 (2002)
20. Noller, H.F., et al.: Secondary structure model for 23S ribosomal RNA. Nucleic Acids Res. 9(22), 6167–6189 (1981)
21. Rzhetsky, A.: Estimating substitution rates in ribosomal RNA genes. Genetics 141(2), 771–783 (1995)
22. Hofacker, I.L., et al.: Automatic detection of conserved RNA structure elements in complete RNA virus genomes. Nucleic acids research 26(16), 3825–3836 (1998)
23. Knudsen, B., Hein, J.: RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. Bioinformatics 15(6), 446–454 (1999)

24. Rivas, E., et al.: Computational identification of noncoding RNAs in E. coli by comparative genomics. Current biology 11(17), 1369–1373 (2001)
25. di Bernardo, D., Down, T., Hubbard, T.: ddbrna: detection of conserved secondary structures in multiple alignments. Bioinformatics 19(13), 1606–1611 (2003)
26. Coventry, A., Kleitman, D.J., Berger, B.: MSARI: multiple sequence alignments for statistical detection of RNA secondary structure. Proceedings of the National Academy of Sciences of the United States of America 101(33), 12102–12107 (2004)
27. Washietl, S., et al.: Mapping of conserved RNA secondary structures predicts thousands of functional noncoding RNAs in the human genome. Nature biotechnology 23(11), 1383–1390 (2005)
28. Pedersen, J.S., et al.: Identification and classification of conserved RNA secondary structures in the human genome. PLoS computational biology 2(4), e33 (2006)
29. Dutheil, J., et al.: A model-based approach for detecting coevolving positions in a molecule. Molecular biology and evolution 22(9), 1919–1928 (2005)
30. Felsenstein, J.: Evolutionary trees from DNA sequences: a maximum likelihood approach. J. Mol. Evol. 17(6), 368–376 (1981)
31. Ranwez, V., Gascuel, O.: Quartet-Based Phylogenetic Inference: Improvements and Limits. Molecular biology and evolution 18(6), 1103–1116 (2001)
32. Atchley, W.R., et al.: Correlations among amino acid sites in bHLH protein domains: an information theoretic analysis. Molecular biology and evolution 17(1), 164–178 (2000)
33. Tillier, E.R., Lui, T.W.: Using multiple interdependency to separate functional from phylogenetic correlations in protein alignments. Bioinformatics 19(6), 750–755 (2003)
34. Comparative RNA Website, `http://www.rna.ccbb.utexas.edu/`
35. PHYLIP, `http://evolution.genetics.washington.edu/phylip.html`
36. Gautheret, D., Damberger, S.H., Gutell, R.R.: Identification of base-triples in RNA using comparative sequence analysis. J. Mol. Biol. 248(1), 27–43 (1995)
37. Gutell, R.R.: Collection of small subunit (16S- and 16S-like) ribosomal RNA structures. Nucleic Acids Res. 21(13), 3051–3054 (1993)
38. Woese, C.R., et al.: Detailed analysis of the higher-order structure of 16S-like ribosomal ribonucleic acids. Microbiol. Rev. 47(4), 621–669 (1983)
39. Patterson, D.A., Hennessy, J.L.: Computer architecture: a quantitative approach, vol. xxviii, 594, p. 160. Morgan Kaufman Publishers, San Mateo (1990)

# What Makes Scientific Workflows Scientific?

Bertram Ludäscher

University of California, Davis
ludaesch@ucdavis.edu

A *scientific workflow* is the description of a process for accomplishing a scientific objective, usually expressed in terms of tasks and their dependencies [5]. While workflows have a long history in the database community as well as in business process modeling (where they are also known as *business workflows*), and despite some early works on scientific workflows [3,10], the area has only recently begun to fully flourish (e.g., see [1,2,9,7,4,11]). Similar to scientific data management which has different characteristics from traditional business data management [8], scientific workflows exhibit new challenges and opportunities that distinguish them from business workflows. We present an overview of these challenges and opportunities, covering a number of issues such as different models of computation, scalable data and process management, and data provenance and lineage handling in scientific workflows.

## References

1. Fox, G.C., Gannon, D. (eds.): Concurrency and Computation: Practice and Experience. Special Issue: Workflow in Grid Systems, vol. 18(10). Wiley & Sons, Chichester (2006)
2. Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., Goble, C., Livny, M., Moreau, L., Myers, J.: Examining the Challenges of Scientific Workflows. Computer 40(12), 24–32 (2007)
3. Ioannidis, Y.E., Livny, M., Gupta, S., Ponnekanti, N.: ZOO: A Desktop Experiment Management Environment. In: VLDB, pp. 274–285 (1996)
4. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J., Zhao, Y.: Scientific Workflow Management and the Kepler System. Concurrency and Computation: Practice & Experience 18(10), 1039–1065 (2006)
5. Ludäscher, B., Bowers, S., McPhillips, T.: Scientific Workflows. In: Özsu, M.T., Liu, L. (eds.) Encyclopedia of Database Systems. Springer, Heidelberg (to appear, 2009)
6. Ludäscher, B., Goble, C. (eds.): ACM SIGMOD Record: Special Issue on Scientific Workflows, vol. 34(3) (September 2005)
7. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20(17) (2004)
8. Shoshani, A., Olken, F., Wong, H.K.T.: Characteristics of Scientific Databases. In: VLDB, pp. 147–160. Morgan Kaufmann, San Francisco (1984)
9. Taylor, I., Deelman, E., Gannon, D., Shields, M. (eds.): Workflows for e-Science: Scientific Workflows for Grids. Springer, Heidelberg (2007)
10. Wainer, J., Weske, M., Vossen, G., Medeiros, C.B.: Scientific Workflow Systems. In: Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems: State of the Art and Future Directions (1996)
11. Yu, J., Buyya, R.: A Taxonomy of Scientific Workflow Systems for Grid Computing. In: Ludäscher and Goble [6]

# Enabling Ad Hoc Queries over Low-Level Scientific Data Sets

David Chiu and Gagan Agrawal

Department of Computer Science and Engineering
The Ohio State University, Columbus, OH 43210, USA
{chiud,agrawal}@cse.ohio-state.edu

**Abstract.** Technological success has ushered in massive amounts of data for scientific analysis. To enable effective utilization of these data sets for all classes of users, supporting intuitive data access and manipulation interfaces is crucial. This paper describes an autonomous scientific workflow system that enables high-level, natural language based, queries over low-level data sets. Our technique involves a combination of natural language processing, metadata indexing, and a semantically-aware workflow composition engine which dynamically constructs workflows for answering queries based on service and data availability. A specific contribution of this work is a metadata registration scheme that allows for a unified index of heterogeneous metadata formats and service annotations. Our approach thus avoids a standardized format for storing all data sets or the implementation of a federated, mediator-based, querying framework. We have evaluated our system using a case study from the geospatial domain to show functional results. Our evaluation supports the potential benefits which our approach can offer to scientific workflow systems and other domain-specific, data intensive applications.

## 1 Introduction

From novel simulations to on-site sensors, advancements in scientific technology have sparked a rapid growth in the deployment of data sources. Vast numbers of low-level data sets, as a result, are persistently stored on distributed disks for access, analysis, and transformation by various classes of users. Managing these low-level data sets on distributed file systems for intuitive user access requires significant consideration towards novel designs in indexing, querying, and integration. At the same time, processes and tools from which the user accesses and manipulates these data sets need to be high-level, if not transparent and automatic. As a result, efforts towards realizing process interoperability and standardized invocation have resulted in the emergence of service-oriented architectures. However, user queries often require a composition of services in able to derive results. And while scientific workflow management systems [1,10,19,22] have made tremendous strides toward scheduling and execution of dependent services, workflows are still largely composed by the user. For scientists and experts, this approach is often sufficient. But in a time when users are becoming more query- and goal-oriented, solutions which leverage effective use of domain information would require significantly less effort.

In this paper we describe a scientific workflow system which enables high-level queries over low-level data sets. Our system, empowered by a nuanced semantics framework, constructs and executes workflow plans automatically for the user. The approach is motivated by the following observations:

– *The aforementioned trend towards service-oriented solutions in scientific computing.* The data grid community, for instance, has benefitted greatly from borrowing web service standards for interoperability through the Open Grid Service Architecture (OGSA) initiative [13].
– *The trend towards the specification of metadata standards in various scientific domains.* These standards allow for clarity in both user and machine interpretability of otherwise cryptic data sets. However, in many sciences, metadata formats are often heterogeneous, and a unified method to index similar information is lacking, specifically for the purposes of workflow composition.

In our system, data sets are required to be *registered* into an index of metadata information, e.g., time, creator, data quality, etc. A simple domain ontology is superimposed across the available data sets and services for enabling workflow planning. Although traditional database and data integration methods (such as the use of federated databases [24] or mediator-based systems [14,25]) can be applied, our approach does not require a standardized format for storing data sets or the implementation of a complex mediator-based querying framework. Our system combines workflow composition with machine-interpretable metadata, a domain ontology, and a natural language interface to offer simple and intuitive ways for querying a variety of data sets stored in their original low-level formats.

Our experimental evaluation is driven through a case study in the geospatial domain. In computing environments with small numbers of data sets, we show that the benefits of our index-enabled workflow planner is unequivocally apparent. Moreover, the scalability of these benefits are easily observable for larger numbers of indices and data sets. Overall, we show that workflows can be composed efficiently using data sets described in disparate metadata formats.

The remainder of this paper is organized as follows: An overview of our system is presented in Section 2. Specifications of majors components in our system are discussed in Section 3. Section 3.1 describes the query decomposition process, followed by metadata registration in Section 3.2. The workflow planning algorithm is presented in Section 3.3. Section 4 explains the results of our performance evaluation on a case study in the geospatial domain. We compare our work with related efforts in Section 5, and finally conclude in Section 6.

## 2 System Overview

The system architecture, shown in Figure 1, consists of four independent layers. The design and implementation of each layer can be superseded without affecting

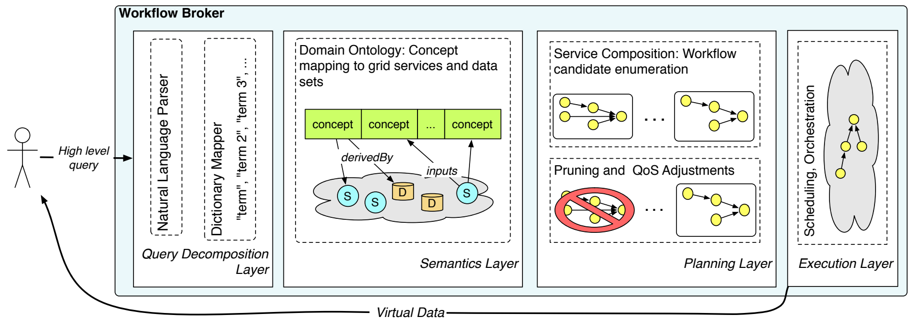

Workflow Broker
High level query
Natural Language Parser
Dictionary Mapper
"term", "term 2", "term 3", ...
Query Decomposition Layer
Domain Ontology: Concept mapping to grid services and data sets
concept
concept
...
concept
derivedBy
inputs
S
D
Semantics Layer
Service Composition: Workflow candidate enumeration
Pruning and QoS Adjustments
Planning Layer
Scheduling, Orchestration
Execution Layer
Virtual Data

**Fig. 1.** System Architecture

the others as long as it subscribes to some system-specified protocols. From a high-level perspective, our system can be viewed as a *workflow broker:* as users submit queries to the system, the broker plans and executes the workflows involved in deriving the desired virtual data while hiding complexities such as workflow composition and domain knowledge from the user.

The user specifies queries through the broker's *Query Decomposition Layer*. This layer decomposes a natural language-based query into keywords and grammatical dependencies using a natural language parser. The parsed set of keywords is then mapped to concepts within the domain ontology specified in the next layer. The *Semantics Layer* maintains an active list of available services, data sets, and their metadata. While the Web Service Description Language (WSDL) [6] is the international standard for web service description, scientific data sets often lack a singular metadata format. For instance, in the geospatial domain alone, CSDGM (Content Standard for Digital Geospatial Metadata) [12] is adopted in the United States. Elsewhere, Europe and Australia have proposed similar metadata standards. More important, XML essentially opens the possibility for any user to define any descriptive data annotation, at any time. But while their formats differ in specification, the information captured is similar: dates of relevance, spatial region, data quality, etc. In the next section, we discuss ways our system deals with heterogeneous metadata.

Although metadata is imperative for providing descriptions for data sets and services, a higher level descriptor is also needed to define the relationships between the available data and services to concepts within some scientific domain. These relationships help facilitate planning algorithms for workflow composition. For example, there is a need for the system to understand how "water levels" are derived using some existing data sets, services, or combinations of both. We specify this description through a simple ontology, a directed graph with the following requirements:

– The ontology consists of three disjoint sets (classes) $C$, $S$, and $D$ representing the set of domain concepts, the set of available services known to the system, and the known domain-specific data types, respectively.
– Two types of directed edges (relations) exist: concepts may be $derivedFrom$ data or service nodes and a service $inputsFrom$ concepts.

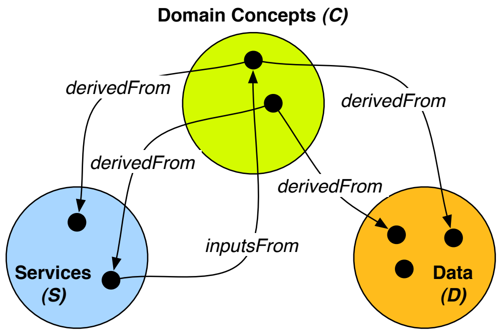


**Fig. 2.** Ontological Structure for Domain Description

This ontological definition, shown in Figure [2], simplifies the effort to indicate which services and data types are responsible for deriving specific domain concepts.

Next, the *Planning Layer* assumes that the ontology and metadata index are in place and defined. The planning algorithm, discussed in detail in the following section, relies heavily on the Semantics Layer. In essence, the planner enumerates workflows to answer any particular query through traversals of the domain ontology. The existence of needed services and data sets is identified by the metadata index. This layer sends a set of workflows all capable of answering the user query to the *Execution Layer* for processing, and the resulting virtual data is finally returned back to the user.

In this paper we focus mainly on the Query Decomposition and Semantics Layers. While the workflow enumeration algorithm in the Planning Layer is also described, details on the algorithm's cost-based pruning mechanism, QoS adaptation, and robustness over distributed environments are discussed in our other publications [8,7].

## 3 Technical Details

This section focuses on the descriptions of the major components involved in supporting queries, semantics, and planning. We lead into the discussion of the technical specifications of each system component through a simple working example query.

```
‘‘return water level from station=32125 on 10/31/2008’’
```

### 3.1 Query Decomposition

The first objective of our system is to process user queries in the form of natural language. The job of the Query Decomposition Layer is to extract relevant elements from the user query. These elements, including the user's desiderata

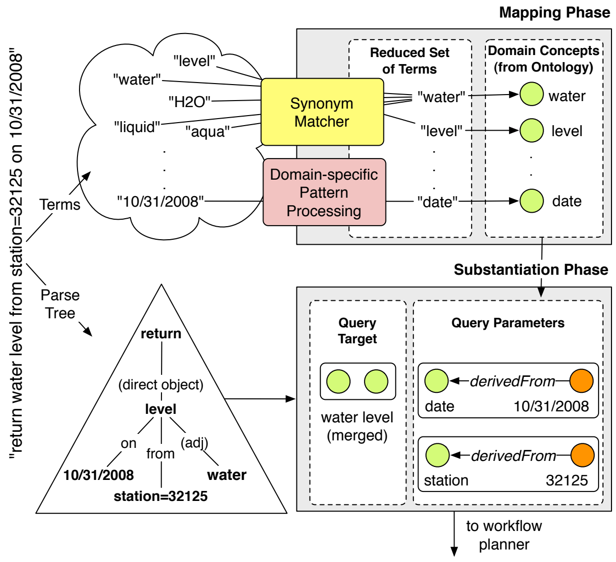


**Fig. 3.** Query Decomposition Process

and other query attributes, are mapped to domain concepts specified in the Semantics Layer's ontology. Thus, these two layers in the system architecture are tightly linked. Shown in Figure 3, the decomposition process is two-phased.

In the Mapping Phase, StanfordNLP [17] is initially employed to output a list of terms and a parse tree from the query. The list of extracted query terms is then filtered through a stop list to remove insignificant terms. This filtered set is further reduced using a synonym matcher provided through WordNet libraries [11]. The resulting term set is finally mapped to individual domain concepts from the ontology. These terms, however, can also take on meaning by their patterns. In our example, "10/31/2008" should be mapped to the domain concept, *date*. A pattern-to-concept matcher, for this reason, was also implemented using regular expressions. But since only certain patterns can be anticipated, some querying guidelines must be set. For instance, dates must be specified in the *mm/dd/yyyy* format, time as *hh:mm:ss*, coordinates as *(lat, long)*, etc. Additionally, values can also be given directly to concepts using a *concept=value* string, as seen for assigning 32125 to *station* in our query.

Upon receiving the set of relevant concepts from the previous phase, the Substantiation Phase involves identifying the user's desired concept as well as assigning the given values to concepts. First, from the given parse tree, concepts are merged with their descriptors. In our example, since "water" describes the term "level", their respective domain concepts are merged. The pattern matcher from the previous phase can be reused to substantiate given values to concepts, resulting in the relations *(date derivedFrom 10/31/2008)* and *(station derivedFrom 32125)*. These query parameter substantiations is stored as a hashset, $Q[\ldots] = Q[date] \rightarrow \{10/31/2008\}$ and $Q[station] \rightarrow \{32125\}$. This set of query parameters is essential for identifying accurate data sets in the workflow planning phase. Query parameters and the target concept, are sent as input to the workflow planning algorithm in the Planning Layer of the system.

We take prudence in discussing our query parser as not to overstate its functionalities. Our parser undoubtedly lacks a wealth of established natural language query processing features (see Section 5), for it was implemented ad hoc for interfacing with our specific domain ontology. We argue that, while related research in this area can certainly be leveraged, the parser itself is ancillary to meeting the system's overall goals of automatic workflow planning and beyond the current scope of this work. Nonetheless, incorrectly parsed queries should be dealt with. Currently, with the benefit of the ontology, the system can deduce the immediate data that users must provide as long as the target concept is determined. The user can then enter the required data into a form for querying.

## 3.2 Metadata Registration

Because workflow planning is a necessary overhead, the existence of data sets (and services) must be identified quickly. Our goal, then, is to provide fast data identification. On one hand, we have the challenge of supplying useful domain knowledge to the workflow planner, and on the other, we have a plethora of pre-existing database/metadata management technologies that can be leveraged. The result is to utilize an underlying database to store and index domain-specific elements and, with the advantage of fast indices, the overhead of data identification for workflow planning can be optimized. For each data set, its indexed domain concepts can be drawn from an accompanying metadata file. However, metadata formats for describing scientific data sets can vary. There exists, for instance, multiple annotation formats from just within the geospatial community. But while their structures differ, the descriptors are similar, storing essential information (data quality, dates, spatial coverage, and so on) pertaining to specific data sets.

Domain experts initialize the system with the following[1]: (i) $\Upsilon = \{\upsilon_1, \ldots, \upsilon_n\}$, a set of XML Schema or Data Type Definitions (DTD) which defines the supported metadata formats used for validation. (ii) $C_{idx}$, a set of domain concepts that the system should index, and (iii) $xpath(\upsilon, c) : (\upsilon \in \Upsilon \wedge c \in C_{idx})$, For each indexed concept and schema, an XPath query [5] that is used to access the indexed value for concept $c$ from a given the metadata document corresponding to schema $\upsilon$. Once in place, domain users should be able to upload and not only share new data sets, but to also make it available for answering high-level queries. To *register* new data sets with the system, users can invoke the Data Registration algorithm. This procedure, shown in Algorithm 1, takes three inputs: a data file $d$, its metadata file $meta_d$, and an optional keyword array, $K[\ldots]$ that describes $d$, and an optional schema for validating $meta_d$, $\upsilon_d$. With the benefit of the term-to-concept mapper described in Section 3.1, the domain concept to which this data set derives can be computed. Optionally, but not shown, the

[1] Our implementation assumes that metadata is defined in XML.

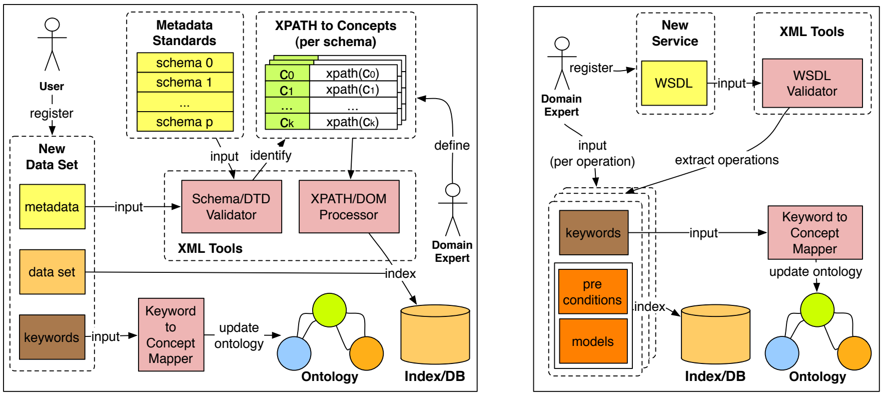
**Algorithm 1.** registerData($d$, $meta_d$[, $K$, $\upsilon_d$])

```
1: /* identify and validate metadata */
2: if $\upsilon_d \in \Upsilon \wedge \upsilon_d$.validate($meta_d$) = true then
3:     $\delta \leftarrow \upsilon_d$ /* input schema checks out */
4: else
5:     for all $\upsilon \in \Upsilon$ do
6:         if $\upsilon$.validate($meta_d$) = true then
7:             $\delta \leftarrow \upsilon$ /* $\delta$ holds the corresponding schema */
8:         end if
9:     end for
10: end if
11: $c_K \leftarrow ConceptMapper$.map($K$) /* solve for concept derived by $d$ */
12: $d_K \leftarrow c_K \cdot$ "type"
13: if $\nexists\ d_K \in Ontology.D$ then
14:     $Ontology.D \leftarrow Ontology.D \cup \{d_K\}$
15:     $Ontology.Edges \leftarrow Ontology.Edges \cup \{(c_K, derivedFrom, d_K)\}$
16: end if
17: /* build database record */
18: $R \leftarrow (datatype = d_K)$
19: for all $c \in C_{idx}$ do
20:     $v \leftarrow meta_d$.extract($xpath(\delta, c)$)
21:     $R \leftarrow record \cup (c = v)$ /* concatenate record */
22: end for
23: $DB$.insert($R, d$)
```

user could select concepts directly from the ontology to describe $d$'s type instead of providing $K[\ldots]$. To avoid confusion of schema validity and versioning, we emphasize here that the set of valid schemas, $\Upsilon$, should only be managed by domain experts or administrators. That is, although users may potentially discover new metadata schemas, our system cannot allow them to update $\Upsilon$ directly.

Algorithm 1 starts by identifying the type of metadata, $\delta$, prescribed by the user via validating $meta_d$ against the set of schemas, or directly against the user provided schema, $\upsilon_d$ (Lines 2-10). Next, the domain concept that is represented by $K[\ldots]$ is solved for on Line 11. On Line 12, $d_K$ is assigned the name representing the type of data in the ontology, where $c_K$ is the matched concept and $\cdot$ denotes string concatenation. If necessary, $d_K$ is added into the ontology's data type class, $D$, and an edge from $c_K$ to $d_K$ is also established (Lines 13-16). Finally, on Lines 18-23, a record is constructed for eventual insertion into the underlying database. The constructed record, $R$, is inserted into the database with a pointer to the data set, $d$. This process is illustrated in Figure 4(a).

Service registration, depicted in Figure 4(b), is not unlike data registration. First, the service description file (in WSDL [6]) is input. Each declared operation must input domain concepts describing its parameters and output. Again, the user may input keywords defining the concept or map the concepts directly. From concept mapping, the ontology is updated with the respective *derivedFrom* and *inputsFrom* edges. Additionally, preconditions and prediction models for



(a) Data Registration (b) Service Registration

**Fig. 4.** Metadata Registration Process

execution time and output size can also be input in this stage. Preconditions, useful for workflow planning, and prediction models, required for QoS adaptation, are both indexed per service operation.

## 3.3 Workflow Planning

Often in practice, scientific tasks are composed of disparate processes chained together to produce some desired values. Although workflows are rooted in business processes, their structures lend well to the realization of complex scientific computing [10,22,1,19]. Workflows can be expressed as directed acyclic graphs where the vertices denote processes/services and data sets and directed edges represent the flow of data. In our framework, we define workflow as follows. Given some arbitrary dataset $D$ and a set of services $S$ a workflow:

$$w = \begin{cases} \epsilon \\ d \\ (s, P_s) \end{cases}$$

such that terminals $\epsilon$ and $d \in D$ denote a null workflow and a data instance respectively. Nonterminal $(s, P_s) \in S$ is a tuple where $s$ denotes a service operation with an ordered parameter list $P_s = (p_1, \ldots, p_k)$ and each $p_i$ is itself a workflow. In other words, a workflow is a tuple which either contains a single data instance or a service whose parameters are, recursively, (sub)workflows.

**Workflow Enumeration Algorithm.** Given some query $q$, the goal of workflow planning algorithm is to enumerate a list of workflows $W_q = (w_1, \ldots, w_n)$ capable of answering $q$ from the available services and data sets. The execution of each $w_i \in W_q$ is carried out, if needed, by an order determined by cost or

QoS parameters. Thus, upon workflow execution failure, the system can persistently attempt alternative, albeit potentially less optimal (with respect to QoS parameters), workflows. Mechanisms for assigning cost to workflows against QoS constraints, however, are out of the scope for this paper.

Domain concept derivation is the goal behind constructing each workflow. Thus, our algorithm, WFEnum, relies heavily on the metadata and semantics provided in the *Semantics Layer*. Recall from Section 3.1 that the Query Decomposition component outputs the query's target concept, $t$, and a hashed set of query parameters, $Q[\ldots]$ (such that $Q[concept] \rightarrow \{val_1, val_2, \ldots\}$). The WFEnum algorithm takes both $t$ and $Q[\ldots]$ as input, and outputs a list $W$ of distinct workflows that are capable of returning the desiderata for the target concept.

WFEnum, shown in Algorithm 2, begins by retrieving all $d \in D$ (types of data registered in the ontology) from which the target concept, $t$, can be derived. On Line 2, a statically accessible array, $W'[\ldots]$, is used for storing overlapping workflows to save redundant recursive calls in the later half of the algorithm. The workflows are memoized on a hash value of their target concept and parameter list. On Line 5, a set of indexed concepts, $C_{idx}$, is identified for each data type, and checked against the parsed user specified values in the query. To perform this check, if the set difference between the registered concepts, $C_{idx}$, and the query parameters, $Q[\ldots]$, is nonempty, then the user clearly did not provide enough information to plan the workflow unambiguously. On Lines 7-11, if all index registered concepts are substantiated by elements within $Q[\ldots]$, a database query is designed to retrieve the relevant data sets. For each indexed concept $c$, its *(concept=value)* pair, $(c = Q[c])$ is concatenated ($AND$'d) to the query's conditional clause. On Lines 12-15, the constructed query is executed and each returned file record, $f$, is an independent file-based workflow deriving $t$.

The latter half of the algorithm deals with concept derivation via service calls. From the ontology, a set of relevant service operations, $\Lambda_{srvc}$ is retrieved for deriving $t$. For each operation, $op$, there may exist multiple ways to plan for its execution because each of its parameters, $p$ , is a subproblem. Therefore, workflows pertaining to each parameter $p$ must first be solved with its own target concept, *p.target* and own subset of relevant query parameters $Q_p[\ldots]$. While *p.target* is easy to identify from following the *inputsFrom* links belonging to $op$ in the ontology, the forwarding of $Q_p[\ldots]$ requires a bit more effort. Looking past Lines 25-31 for now, this query parameter forwarding process is discussed in detail in Section 3.3.

Once the $Q_p[\ldots]$ is forwarded appropriately, the recursive call can be made for each parameter, or, if the call is superfluous, the set of workflows can be retrieved directly (Line 32-36). In either case the results are stored in $W_p$, and the combination of these parameter workflows in $W_p$ is established through a cartesian product of its derived parameters (Line 37). For instance, consider a service workflow with two parameters of concepts $a$ and $b$: $(op, (a, b))$. Assume that target concepts $a$ is derived using workflows $W_a = (w_1^a, w_2^a)$ and $b$ can only be derived with a single workflow $W_b = (w_1^b)$. The distinct parameter

**Algorithm 2.** WFEnum($t$, $Q[\ldots]$)

```
1:  $W \leftarrow ()$
2:  global $W'[\ldots]$ /* static table for memoization */
3:  $\Lambda_{data} \leftarrow Ontology.$derivedFrom($D$, $t$)
4:  **for all** $d \in \Lambda_{data}$ **do**
5:      $C_{idx} \leftarrow d.$getIndexConcepts()
6:      /* user-given values enough to substantiate indexed concepts */
7:      **if** ($Q$.concepts() $-$ $C_{idx}$) $= \{\}$ **then**
8:          $cond \leftarrow (datatype = d)$
9:          **for all** $c \in C_{idx}$ **do**
10:             $cond \leftarrow cond \wedge (c = Q[c])$ /* concatenate new condition */
11:         **end for**
12:         $F \leftarrow \sigma_{<cond>}(datasets)$ /* select files satisfying $cond$ */
13:         **for all** $f \in F$ **do**
14:             $W \leftarrow (W, (f))$
15:         **end for**
16:     **end if**
17: **end for**
18:
19: $\Lambda_{srvc} \leftarrow Ontology.$derivedFrom($S$, $t$)
20: **for all** $op \in \Lambda_{srvc}$ **do**
21:     $\Pi_{op} \leftarrow op.$getPreconditions();
22:     $P_{op} \leftarrow op.$getParameters()
23:     $W_{op} \leftarrow ()$
24:     **for all** $p \in P_{op}$ **do**
25:         /* forward query parameters s.t. preconditions are not violated */
26:         $Q_p[\ldots] \leftarrow Q[\ldots]$
27:         **for all** $(concept, value) \in Q_p[\ldots]$ **do**
28:             **if** $(concept, value)$.violates($\Pi_{op}$) **then**
29:                 $Q_p[\ldots] \leftarrow Q_p[\ldots] - (concept, value)$
30:             **end if**
31:         **end for**
32:         **if** $\exists W'[h(p.target, Q_p)]$ **then**
33:             $W_p \leftarrow W'[h(p.target, Q_p)]$ /* recursive call is redundant */
34:         **else**
35:             $W_p \leftarrow$ WFEnum($p.target$, $Q_p[\ldots]$) /* recursively invoke for $p$ */
36:         **end if**
37:         $W_{op} \leftarrow W_{op} \times W_p$ /* cartesian product */
38:     **end for**
39:     /* couple parameter list with service operation and concatenate to $W$ */
40:     **for all** $pm \in W_{op}$ **do**
41:         $W \leftarrow (W, (op, pm))$
42:     **end for**
43: **end for**
44: $W'[h(t, Q_p)] \leftarrow W$ /* memoize */
45: return $W$
```

list plans are thus obtained as $W_{op} = W_a \times W_b = ((w_1^a, w_1^b), (w_2^a, w_1^b))$. Each element from $W_{op}$ is a unique parameter list. These lists are coupled with the service operation, *op*, memoized in $W'$ for avoiding redundant recursive calls in the future, and returned in $W$ (Lines 39-45). In our example, the final list of workflows is obtained as $W = ((op, (w_1^a, w_1^b)), (op, (w_2^a, w_1^b)))$.

The returned list, $W$, contain planned workflows capable of answering an original query. Ideally, $W$ should be a queue with the "best" workflows given priority. Mechanisms identifying the "best" workflows to execute, however, depends on the user's preferences. Our previous effort have led to QoS-based cost scoring techniques leveraging on bi-criteria optimization: workflow execution time and result accuracy. Although not shown in this paper, the gist of this effort is to train execution time models and also allow domain experts to input error propagation models per service operation. Our planner, when constructing workflows, invoke the prediction models based on user criteria. Workflows not meeting either constraint are pruned on the a priori principle during the enumeration phase. In the special case of when $W$ is empty, however, a re-examination of pruned workflows is conducted to dynamically adapt to meet these constraints through data reduction techniques. This QoS adaptation scheme is detailed in other publications [8,7].

**Forwarding of Query Parameters.** It was previously noted that planning a service operation is dependent on the initially planning of the operation's parameters. This means that WFEnum must be recursively invoked to plan (sub)workflows for each parameter. Whereas the (sub)target concept is clear to the system from *inputsFrom* relations specified in the ontology, the original query parameters must be forwarded correctly. For instance, consider some service-based workflow, $(op, (L_1, L_2))$ that expects as input two time-sensitive data files: $L_1$ and $L_2$. Let's then consider that *op* makes the following two assumptions: (i) $L_1$ is obtained at an earlier time/date than $L_2$ and (ii) $L_1$ and $L_2$ both represent the same spatial region. Now assume that the user query provides two dates, 10/2/2007 and 12/3/2004 and a location $(x, y)$, that is,

$$Q[\ldots] = \begin{cases} location \rightarrow \{(x, y)\} \\ date \rightarrow \{10/2/2007, 12/3/2004\} \end{cases}$$

To facilitate this distribution, the system allows a set of preconditions, $\Pi_{op}$, to be specified per service operation. All conditions from within $\Pi_{op}$ must be met before allowing the planning/execution of *op* to be valid, or the plan being constructed is otherwise abandoned. In our case, the following preconditions are necessary to capture the above constraints:

$$\Pi_{op} = \begin{cases} L_1.date \preceq L_2.date \\ L_1.location = L_2.location \end{cases}$$

In Lines 25-31, our algorithm forwards the values accordingly down their respective parameter paths guided by the preconditions, and thus implicitly satisfying them. The query parameter sets thus should be distributed differently for the recursively planning of $L_1$ and $L_2$ as follows:

$$Q_{L_1}[\ldots] = \begin{cases} location \rightarrow \{(x, y)\} \\ date \rightarrow \{12/3/2004\} \end{cases} \quad Q_{L_2}[\ldots] = \begin{cases} location \rightarrow \{(x, y)\} \\ date \rightarrow \{10/2/2007\} \end{cases}$$

The recursive planning for each (sub)workflow is respectively supplied with the reduced set of query parameters to identify only those files adhering to preconditions.

## 4 System Evaluation

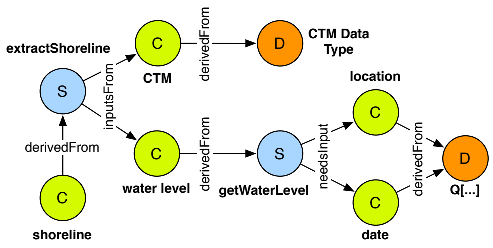
The experiments that we conducted are geared towards exposing two particular aspects of our system: (i) we run a case study from the geospatial domain to display its functionality, including metadata registration, query decomposition, and workflow planning. (ii) We show scalability and performance results of our query enumeration algorithm, particularly focusing on data identification.

**Experimental Case Study.** To present our system from a functional standpoint, we employ an oft-used workflow example from the geospatial domain: shoreline extraction. This application requires a Coastal Terrain Model (CTM) file and water level information at the targeted area and time. CTMs are essentially matrices (from a topographic perspective) where each point represents a discretized land elevation or bathymetry (underwater depth) value in the captured coastal region. To derive the shoreline, and intersection between the effective CTM and a respective water level is computed. Since both CTM and water level data sets are spatiotemporal, our system must not only identify the data sets efficiently, but plan service calls and their dependencies accurately and automatically.

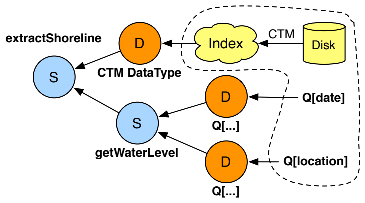
For this example, the system's data index is configured to include only *date* and *location* concepts. In practice however, it would be useful to index additional elements such as resolution/quality, creator, map projection, and others. Next, we provided the system with two metadata schemas, the U.S.-based CSDGM [12] and the Australia and New Zealand standard, ANZMETA [3], which are both publicly available. Finally, XPaths formed from the schemas to index concepts *date* and *location* for both schemas are defined.

Next, CTM files, each coupled with corresponding metadata and keywords $K = \{$ *"CTM", "coastal terrain model", "coastal model"*$\}$, are inserted into the system's registry using the data registration procedure provided in Algorithm 1. In the indexing phase, since we are only interested in the spatiotemporal aspects of the data sets, a single modified B$^x$-Tree [16] is employed as the underlying database index for capturing both *date* and *location*.[2] For the ontology phase,

[2] Jensen et al.'s B$^x$-Tree [16], originally designed for moving objects, is a B+Tree whose keys are the approximate linearizations of time and space of the object via space-filling curves.



(a) Example Ontology after Registration



(b) Shoreline Workflow Structure

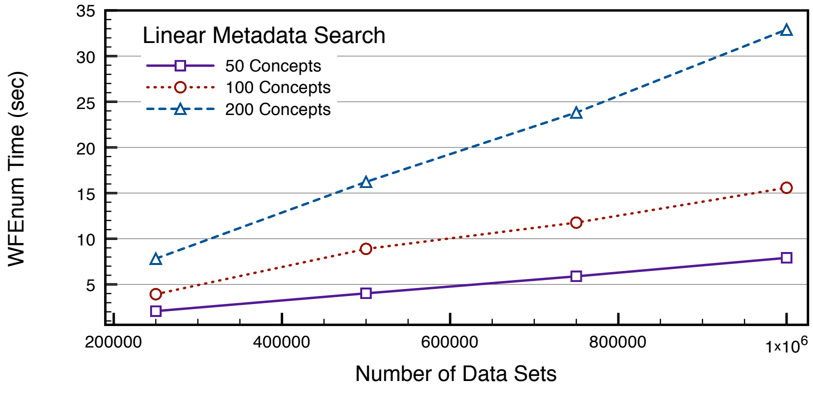
**Fig. 5.** The Shoreline's Involved Ontology and the Derived Workflow

since a *CTM* concept is not yet captured in the domain ontology, the keyword-to-concept mapper will ask the user to either (a) display a list of concepts, or (b) create a new domain concept mapped from keywords $K$. If option (a) is taken, then the user chooses the relevant concept and the incoming data set is registered into the ontology, and $K$ is included the mapper's dictionary for future matches. Subsequent CTM file registrations, when given keywords from $K$, will register automatically under the concept *CTM*. On the service side, two operations are required for registration, shown below as $(op, (c_{p1}, c_{p2}, \ldots, c_{pk}))$, where *op* denotes the service operation name and $c_{pi}$ denotes the domain concept of parameter $i$:

1. *(getWaterLevel, (date, location))*: retrieves the average water level reading on the given date from a coastal gauging station closest to the given location.
2. *(extractShoreline, (CTM, water level))*: intersects the given CTM with the water level and computes the shoreline.

For sake of simplicity, neither operation requires preconditions and cost prediction models. After metadata registration, the resulting ontology is shown in Figure 5(a), unrolled for clarity. Albeit that there are a multitude of more nodes in a practical system, it is easy to see how the WFEnum algorithm would plan for shoreline workflows. By traversing from the targeted concept, *shoreline*, and visiting all reachable nodes, the workflow structure is a reduction of *shoreline*'s reachability subgraph with a reversal of the edges and a removal of intermediate concept nodes. The abstract workflow shown in Figure 5(b) is the general structure of all plannable workflows. In this particular example, WFEnum will enumerate more than one workflow candidate only if multiple CTM files (perhaps of disparate resolutions) are registered in the index at the queried location and time.

**Performance Evaluation.** Our system is distributed by nature, and therefore, our testbed is structured as follows. The workflow planner, including metadata indices and the query parser, is deployed onto a Linux machine running a Pentium 4 3.00Ghz Dual Core with 1GB of RAM. The geospatial processes are deployed as web services on a separate server located across the Ohio State University campus at the Department of Civil and Environmental Engineering and Geodetic Science.



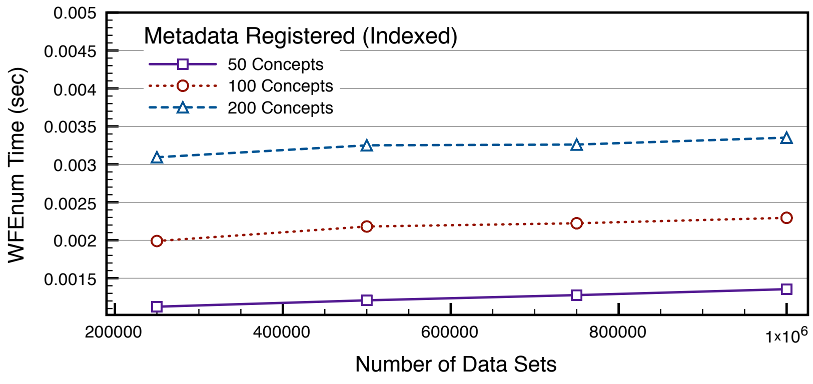


**Fig. 6.** Workflow Planning Times with Increasing Data Sets and Concept Indices

CTM data sets, while indexed on the workflow planner node, are actually housed on a file server across state, at the Kent State University campus.

In the first experiment, we are interested in the runtime of WFEnum with and without the benefit of metadata registration when scaled to increasing amounts of data files and concepts needing indexed (thus resulting in both larger index structures and a larger number of indices). Shown in Figure 6 (top), the linear search version consumes significant amounts of time, whereas its counterpart (bottom) consumes mere milliseconds for composing the same workflow plan. Also, because dealing with multiple concept indices is a linear function, its integration into linear search produces drastic slowdowns. And although the slowdown can also be observed for the indexed runtime, they are of negligible amounts.

Once the shoreline extraction workflow has finished planning, its execution is then carried out by our system. As seen in Figure 7, the workflow's execution time is heavily dependent on the CTM file size. If we juxtaposed Figure 6 with Figure 7, the importance of minimizing planning time becomes clear. Especially for smaller CTM files, the cases when planning times dominate execution

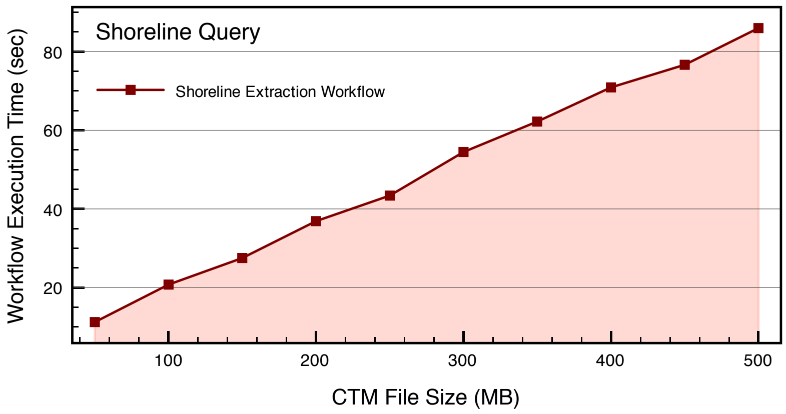


**Fig. 7.** Shoreline Workflow Execution Times

times should be avoided, and metadata indexing decreases the likelihood for this potential.

## 5 Related Efforts

The need for metadata and semantics has long been addressed by such initiatives as the plethora of XML-based technologies, including Resource Description Framework (RDF) and its complement, the Web Ontology Language (OWL) [20,9]. These standards have opened up support to allow anyone to specify human and machine interpretable descriptions for any type of data. In our system, we indeed employ RDF+OWL to formalize a general ontology which describes the relationships between concepts and resources (data sets and services). This resource description is imperative to our system, as it semantically drives the workflow planner.

Parsing correctness, disambiguation, and schema mapping are well-known problems in natural language querying. Stanford's Natural Language Parser [17] and dictionary API's provided by WordNet [11] are often employed in systems providing natural language support, including our own. In the direction of querying structured information, ample research has been developed for addressing the issues with translating natural language translation to structured queries [2,18]. We concede that our parser is lacking such relevant technologies for handling the age-old challenges of disambiguation, mapping, etc. Undertaking the implementation of these features is currently beyond the scope of this work.

Research in high performance scientific data management has produced such systems as the Scientific Data Manager (SDM), which employs the Meta-data Management System (MDMS) [21]. SDM provides a programming model and abstracts low-level parallel I/O operations for complex scientific processing. While MDMS uses a database for metadata storage, the metadata itself is specific to the scientific process at hand, containing information on execution (e.g., access patterns, problem size, data types, file offsets, etc). This metadata is used by SDM to optimize the runtime of these parallel processes. Another system, San Diego Supercomputing Center's Storage Resource Broker (SRB) [4], seeks to store massive volumes of data sets split across clusters or nodes within heterogenous environments. SRB allows parallel and transparent data access by offering a simplified API to users which hides complexities such as merging data sets, allowing restricted access, etc. Compared to our system, there is a fundamental difference in functionality. Ours provides a way to store heterogeneous metadata specific to scientific domains inside a database, and that the metadata are invoked not for process optimization, but for data identification purposes for automatic workflow planning.

Ways to handle the heterogeneity of metadata have prompted many works on metadata cataloguing and management. Particularly, in the volatile grid computing environment, data sources are abundant and metadata sources are ever-changing. Metadata Catalog Service (MCS) [26] and Artemis [28] are collaborative components used to access and query repositories based on metadata attributes. MCS is a self-sufficient catalog which stores information on data sets. Its counterpart Artemis, on the other hand, can be used to integrate many versions of MCS for answering interactive queries. Their interface takes users through a list of questions guided by a domain ontology to formulate a query. The planned query is then sent to the Artemis mediator to search for relevant items in the MCS instances. While the MCS and Artemis is somewhat tantamount to our metadata registration and automatic query formulation processes, our systems differ in the following ways. (i) Ours not only facilitates accurate data identification based on metadata querying, but also combining these data items with similarly registered services to compose workflows. (ii) Although both systems allow higher level querying frameworks, our approach is enabled through natural language and keyword mapping of domain ontology concepts.

Workflow management systems have also gained momentum in the wake of managing complex, user-driven, scientific computations in the form of service composition. By itself, service composition have become prevalent enough to warrant such industrial standards as the WSBPEL (Web Service Business Process Execution Language) [30] to describe the orchestration of service execution. Implementations of WSBPEL engines have already sprawled into realms of proprietary and open-source communities, an auspicious indication of the high optimism for the movement toward service-oriented workflow solutions. In fact, many efforts towards scientific workflow composition have already been developed. Notable systems such as Askalon [29], Taverna [22], and Kepler [1] have evolved into grid- and service-oriented systems. These systems typically allow domain experts to define static workflows through a user-friendly interface, and map the component processes to known services. Pegasus [10,15] allows users to compose workflows potentially with thousands of nodes using abstract workflow templates. But while these systems alleviate users' efforts for composition, our system proposes automatic workflow planning based on available metadata to elude user-based composition altogether.

In the direction of automatic workflow composition, Traverso et al. discussed the importance of exploiting semantic and ontological information for automating service composition [27]. Their approach generates automata-based "plans," which can then be translated into WSBPEL processes. The goals and requirements for these plans, however, must be expressed in a formal language, which may be cryptic for the average user. Other automatic planning-based systems, such as Sword [23] and SHOP2 [31], also require similar complexity in expressing workflows. While the overall objectives of these systems are tantamount to those of our own, our directions are quite different. In an age when scientific data sets are ubiquitous and when machine- and human-interpretable descriptions are imperative, we are invariably deluged with high-volumes of heterogeneous data sets and metadata. To the best of our knowledge, the registration and exploitation of domain-specific metadata to automatically compose workflows for answering natural language queries is a novel approach in this area.

## 6 Conclusion and Future Work

In this paper we have presented a system which supports simplified querying over low-level scientific datasets. This process is enabled through a combination of effective indexing over metadata information, a system and domain specific ontology, and a workflow planning algorithm capable of alleviating all tiers of users of the difficulties one may experience through dealing with the complexities of scientific data. Our system presents a new direction for users, from novice to expert, to share data sets and services. The metadata, which comes coupled with scientific data sets, is indexed by our system and exploited to automatically compose workflows in answering high level queries without the need for common users to understand complex domain semantics.

As evidenced by our experiments, a case can be made for supporting metadata registration and indexing in an automatic workflow management system. In our case study alone, comparing the overhead of workflow planning between linear search and index-based data identification methods, speedups are easily observed even for small numbers of data sets. Further, on the medium scale of searching through $1 \times 10^6$ data sets, it clearly becomes counterproductive to rely on linear metadata search methods, as it potentially takes longer to plan workflows than to execute them. As evidenced, this scalability issue is easily mitigated with an indexed approach, whose planning time remains negligible for the evaluated sizes of data sets.

Although our system claims to support natural language queries, it is, admittedly, far from complete. For instance, mapping sophisticated query elements to supporting range queries and joins is lacking. While this limits querying support significantly, we believe that these details can be realized with more effort spent on providing better models in the relationship between parse trees and the query plan. Nonetheless, in a more holistic sense, these nuances are diminutive against the general role of the system.

## Acknowledgments

This work was supported by NSF grants 0541058, 0619041, and 0833101. The equipment used for the experiments reported here was purchased under the grant 0403342.

## References

1. Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludscher, B., Mock, S.: Kepler: An extensible system for design and execution of scientific workflows (2004)
2. Androutsopoulos, I., Ritchie, G.D., Thanisch, P.: Natural language interfaces to databases–an introduction. Journal of Language Engineering 1(1), 29–81 (1995)
3. ANZLIC. Anzmeta xml document type definition (dtd) for geospatial metadata in australasia (2001)
4. Baru, C., Moore, R., Rajasekar, A., Wan, M.: The sdsc storage resource broker. In: CASCON 1998: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research, p. 5. IBM Press (1998)
5. Berglund, A., Boag, S., Chamberlin, D., Fernndez, M.F., Kay, M., Robie, J., Simon, J.: Xml path language (xpath) 2.0. w3c recommendation (January 23, 2007), http://www.w3.org/tr/xpath20
6. Chinnici, R., Moreau, J.-J., Ryman, A., Weerawarana, S.: Web services description language (wsdl) 2.0. w3c recommendation (June 26, 2007), http://www.w3.org/tr/wsdl20/
7. Chiu, D., Deshpande, S., Agrawal, G., Li, R.: Composing geoinformatics workflows with user preferences. In: GIS 2008: Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems. ACM, New York (2008)
8. Chiu, D., Deshpande, S., Agrawal, G., Li, R.: Cost and accuracy sensitive dynamic workflow composition over grid environments. In: Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (Grid 2008) (2008)
9. Dean, M., Schreiber, G.: Owl web ontology language reference. w3c recommendation (2004)
10. Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A.C., Jacob, J.C., Katz, D.S.: Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming 13(3), 219–237 (2005)
11. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press, Cambridge (1998)
12. FGDC. Metadata ad hoc working group. content standard for digital geospatial metadata (1998)
13. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: An open grid services architecture for distributed systems integration (2002)
14. Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J.: Integrating and Accessing Heterogenous Information Sources in TSIMMIS. In: Proceedings of the AAAI Symposium on Information Gathering (1995)
15. Gil, Y., Ratnakar, V., Deelman, E., Mehta, G., Kim, J.: Wings for pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In: Proceedings of the 19th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI), Vancouver, British Columbia, Canada, July 22-26 (2007)

16. Jensen, C.S., Lin, D., Ooi, B.C.: Query and update efficient b+tree-based indexing of moving objects. In: Proceedings of Very Large Databases (VLDB), pp. 768–779 (2004)
17. Klein, D., Manning, C.D.: Accurate unlexicalized parsing. In: Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp. 423–430 (2003)
18. Li, Y., Yang, H., Jagadish, H.V.: Nalix: an interactive natural language interface for querying xml. In: SIGMOD Conference, pp. 900–902 (2005)
19. Majithia, S., Shields, M.S., Taylor, I.J., Wang, I.: Triana: A Graphical Web Service Composition and Execution Toolkit. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2004), pp. 514–524. IEEE Computer Society, Los Alamitos (2004)
20. Manola, F., Miller, E.: Resource description framework (rdf) primer. w3c recommendation (2004)
21. No, J., Thakur, R., Choudhary, A.: Integrating parallel file i/o and database support for high-performance scientific data management. In: Supercomputing 2000: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM), Washington, DC, USA, p. 57. IEEE Computer Society, Los Alamitos (2000)
22. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20(17), 3045–3054 (2004)
23. Ponnekanti, S.R., Fox, A.: Sword: A developer toolkit for web service composition. In: WWW 2002: Proceedings of the 11th international conference on World Wide Web (2002)
24. Sheth, A., Larson, J.: Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases. ACM Computing Surveys 22(3), 183–236 (1990)
25. Shklar, L., Sheth, A., Kashyap, V., Shah, K.: InfoHarness: Use of Automatically Generated Metadata for Search and Retrieval of Heterogeneous Information. In: Iivari, J., Rossi, M., Lyytinen, K. (eds.) CAiSE 1995. LNCS, vol. 932. Springer, Heidelberg (1995)
26. Singh, G., Bharathi, S., Chervenak, A., Deelman, E., Kesselman, C., Manohar, M., Patil, S., Pearlman, L.: A metadata catalog service for data intensive applications. In: SC 2003: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, p. 33. IEEE Computer Society, Washington (2003)
27. Traverso, P., Pistore, M.: Automated composition of semantic web services into executable processes. In: 3rd International Semantic Web Conference (2004)
28. Tuchinda, R., Thakkar, S., Gil, A., Deelman, E.: Artemis: Integrating scientific data on the grid. In: Proceedings of the 16th Conference on Innovative Applications of Artificial Intelligence (IAAI), pp. 25–29 (2004)
29. Wieczorek, M., Prodan, R., Fahringer, T.: Scheduling of scientific workflows in the askalon grid environment. SIGMOD Rec. 34(3), 56–62 (2005)
30. Web services business process execution language (wsbpel) 2.0, oasis standard
31. Wu, D., Sirin, E., Hendler, J., Nau, D., Parsia, B.: Automatic web services composition using shop2. In: ICAPS 2003: International Conference on Automated Planning and Scheduling (2003)

# Exploring Scientific Workflow Provenance Using Hybrid Queries over Nested Data and Lineage Graphs

Manish Kumar Anand[1], Shawn Bowers[2,3], Timothy McPhillips[2], and Bertram Ludäscher[1,2]

[1] Department of Computer Science, University of California, Davis
[2] UC Davis Genome Center, University of California, Davis
[3] Department of Computer Science, Gonzaga University
{maanand,sbowers,tmcphillips,ludaesch}@ucdavis.edu

**Abstract.** Existing approaches for representing the provenance of scientific workflow runs largely ignore computation models that work over structured data, including XML. Unlike models based on transformation semantics, these computation models often employ update semantics, in which only a portion of an incoming XML stream is modified by each workflow step. Applying conventional provenance approaches to such models results in provenance information that is either too coarse (e.g., stating that one version of an XML document depends entirely on a prior version) or potentially incorrect (e.g., stating that each element of an XML document depends on every element in a prior version). We describe a generic provenance model that naturally represents workflow runs involving processes that work over nested data collections and that employ update semantics. Moreover, we extend current query approaches to support our model, enabling queries to be posed not only over data lineage relationships, but also over versions of nested data structures produced during a workflow run. We show how hybrid queries can be expressed against our model using high-level query constructs and implemented efficiently over relational provenance storage schemes.

## 1 Introduction

Scientific workflow systems (e.g., [15,7,19]) are increasingly used by scientists to design and execute data analysis pipelines and to perform other tool integration tasks. Workflows in these systems often are represented as directed graphs where nodes denote computation steps (e.g., for data acquisition, integration, analysis, or visualization) and edges represent the required dataflow between steps. Systems execute workflow graphs according to various *models of computation* [15], which generally specify how workflow steps should be scheduled and how data should be passed (and managed) between steps. In addition to automating data analyses, scientific workflow systems can capture the detailed provenance of data produced during workflow runs, often by recording the processing steps used to derive data products and the data provided to and generated by each step. Provenance recording capabilities represent a key advantage of scientific workflow technology over more traditional scripting approaches, enabling scientists to more easily understand, reproduce, and verify scientific results [9,4]. However, effectively representing provenance information is complicated by a number of

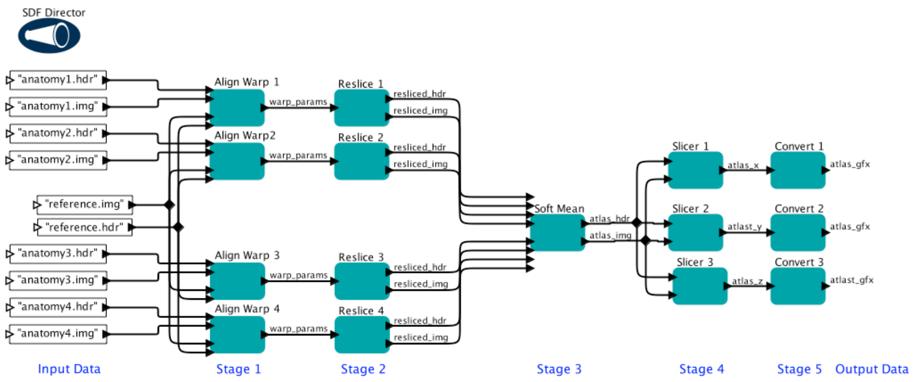


**Fig. 1.** A straightforward Kepler implementation of the first provenance challenge fMRI workflow. Given 3D brain scans (anatomy images) it: (1) compares each image to a reference image to determine "warping" parameters for alignment with the reference; (2) transforms the images according to the parameters; (3) averages the transformed images into an atlas image; (4) produces three different 2D slices of the altas; and (5) converts each 2D slice into a graphical image.

technical challenges as workflows, the models of computation used to enact them, and the structures of the data flowing through them, each become more complex.

**Design Challenges.** Consider the Kepler workflow definition shown in Fig. 1, which is a straightforward implementation of the fMRI image processing pipeline of the first provenance challenge [18] (i.e., directly following the implementation suggested in the challenge). This workflow design hardwires the number of input images (four; see Stages 1–3) and the number of output slices (three; see Stages 4–5). Consequently, performing the same computation on a different number of input images or with additional output slices will require significant modifications to the workflow definition. Simply supplying a different input data set or changing parameter values applied to the workflow is not sufficient. Instead new components (actors) and additional wires must be introduced throughout the workflow. The limitation of this design approach is even more obvious in workflows where the number of data items produced *during* a workflow run is not predetermined. For example, many standard bioinformatics analyses include workflow steps that produce *collections* of output data with indefinite cardinality (e.g., BLAST and maximum parsimony tree inference algorithms [5], among others). Chaining such steps together generally requires data to be grouped into nested collections, and workflow systems often address this need by enabling workflow authors to model data using XML-like data structures. Further, by natively supporting operations over such nested data collections (e.g., [11,20,19,5]), these systems can also yield more generic and reusable workflow designs [16].

**Design via Nested Data Collections.** Fig. 2 illustrates the advantages of employing XML-like data structures in workflows. Shown is a workflow definition with the same intent as that in Fig. 1, but implemented using the *Collection-oriented modeling and*

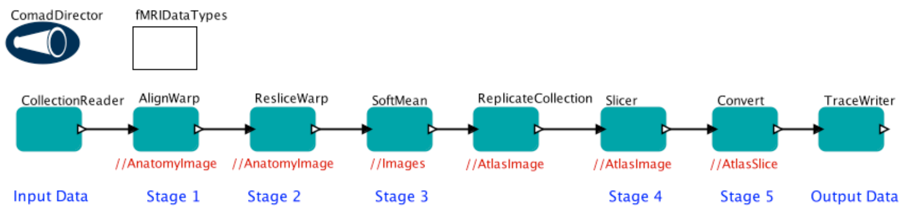


**Fig. 2.** A COMAD design of the fMRI workflow in Kepler, where: (1) CollectionReader is configured with an input XML structure that specifies the input images used; (2) AlignWarp, ResliceWarp, SoftMean, Slicer, and Convert are similar to those of Fig. 1 but work over portions of the XML data stream; and (3) ReplicateCollection is configured to create *n* copies of the resulting SoftMean images, where each copy induces a separate Slicer invocation. COMAD extends Kepler with explicit support for managing data collections, making COMAD actors "collection aware."

*design* (i.e., COMAD) paradigm [16] developed in Kepler.[1] Workflows in COMAD are executed over XML data streams in which actors employ *update semantics* by making modifications to (i.e., updating) portions of the overall XML structure and passing the updated structure to downstream actors. Specifically, actors in COMAD receive fragments of an incoming XML token stream based on their declared *read-scope* parameters (given as XPath expressions in Fig. 2), insert and remove fragments within their matching scopes, and pass the modified stream on to subsequent actors. Actors are executed (or *invoked*) over each corresponding read scope match within an incoming token stream. For instance, the COMAD implementation of the Align Warp actor (Stage 1) is invoked once over each Anatomy Image collection within the workflow input, as shown in Fig. 3a for the first invocation of Align Warp. The workflow of Fig. 2 differs from Fig. 1 in that it can be executed over multiple collections of anatomy images of varying cardinality without requiring modifications to the workflow graph. To have the workflow average five anatomy images, rather than four, the user only needs to add the additional image to the input data set. The COMAD version of the workflow also contains noticeably fewer overall actor occurrences and connections. Other advantages include support for parallel actor execution (where actors are executed concurrently over distinct portions of the overall XML structure) and the ability to easily add and remove actors within a pipeline while minimizing the changes that must be made to the workflow graph [16].

**Provenance Challenges.** While COMAD provides benefits for workflow design and execution, it requires a richer model of provenance than used by systems in which actors correspond to *data transformers* (as in Fig. 1), i.e., where actors (i) treat data as opaque objects (or *tokens*), (ii) produce a set of new output tokens from each set of input tokens they receive, and (iii) assume all input tokens are used to derive all output tokens within a single actor invocation. Provenance management systems tailored to workflows consisting entirely of data transformers (e.g., [2,21,17]) generally store the input and output tokens of each actor invocation, and later use this information to infer (causal) dependencies between data tokens and workflow steps. When applied to

[1] Kepler supports multiple computation models ( *directors*) including standard dataflow models such as Synchronous DataFlow (SDF) and Process Networks (PN) [15].

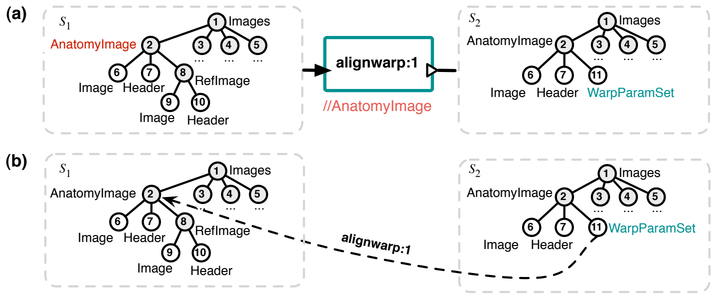


**Fig. 3.** An actor invocation employing *update semantics*: (a) example nested data structures $s_1$ and $s_2$ input to and output by the first invocation of the AlignWarp actor during a run of Fig. 2; and (b) the fine-grain data dependencies of nodes in $s_2$ on nodes in $s_1$ introduced by the invocation

computation models such as COMAD, however, this standard approach to recording and reporting provenance relationships can infer incomplete and even incorrect data and invocation dependencies [3]. For example, Fig. 3b shows the actual data dependencies introduced by the first invocation of the AlignWarp actor; specifically, the object representing the warping parameters (node 11) was derived from each of the image and header objects contained in the AnatomyImage collection matching the read scope of the invocation (node 2). However, if these input and output structures were represented as atomic data objects ($s_1$ and $s_2$), the standard model would infer only a single dependency between the output version ($s_2$) and the input version ($s_1$). Alternatively, if as in COMAD, XML structures are represented as XML token streams, then the standard model would incorrectly infer dependencies between every output and input node.

**Contributions.** We define a model of provenance (based on our prior work [3]) that extends the standard model [17,9] with support for scientific workflows that process XML data and employ update semantics. We also present approaches for querying provenance information based on this model. The model provides a *conceptual* representation of workflow provenance that is used for expressing provenance queries and for interpreting query results. A goal of our approach is to address shortcomings of current approaches for querying provenance information. Specifically, most existing approaches directly expose to users the physical representation of provenance information (e.g., using relational, XML, or RDF schemas) in which users must express provenance queries using associated query languages (e.g., SQL, XQuery, or SPARQL). These approaches are often invconvenient and difficult for users needing to express complex provenance queries, and also limit opportunities for storage and query optimization, since optimization often results in modifications to the underlying provenance representation. As an alternative, we present a *Query Language for Provenance* (QLP; pronounced “clip”) that is designed to be independent of any particular physical representation, and that includes constructs tailored specifically for querying scientific workflow provenance. QLP constructs also allow queries to be expressed over both lineage information and different versions of nested data structures produced by workflow runs. We show that QLP can express common provenance queries and that these queries can be answered efficiently based on translations to standard relational database techniques.

**Outline.** Section 2 describes our provenance model for representing COMAD-style workflow runs that supports nested data and update semantics. Section 3 describes shortcomings of current approaches for querying provenance information, presents the QLP query constructs, and shows that these constructs are expressive enough to formulate common provenance queries. We also briefly describe our implementation of QLP and demonstrate the feasibility of our approach in Section 3. Section 4 discusses related work, and Section 5 summarizes our contributions.

## 2 Models of Provenance

Despite efforts to better understand and even standardize provenance models, e.g., in the Open Provenance Model (OPM) initiative [18], a universally accepted model has yet to emerge. This is due in part to the differences in the underlying models of computation employed by workflow systems. In the following, we develop a number of models of provenance, starting with a basic model capturing the conventional view of scientific workflows as simple task dependency graphs over atomic data and atomic (single invocation) processes, similar to the OPM model (as well as others, e.g., [7]). Next, we extend this basic model to handle computations where a workflow step (i.e., task or process) consists of multiple invocations (or *firings* [14]) over a stream of incoming tokens. The resulting model is a variant of *process networks* [13] that is well-suited for stream processing, and comes with "built-in" pipeline parallelism. A second, and for our purposes crucial extension of the basic provenance model, allows us to handle complex data, i.e., nested data collections, represented in XML. We discuss two variants of this XML-based provenance model, with *copy* and *update* semantics, respectively. Finally, our last extension yields a unified provenance model, incorporating the above features and adding fine-grained dependencies such as in Fig. 3b.

**Basic Model.** The conventional (or *basic*) model of provenance consists of a *trace structure* (or simply a *trace*) $T = (V, E)$ forming an acyclic *flow graph*, where each node in $V = S \cup I$ represents either an (atomic) *data structure* $s \in S$ or a *process invocation* $i \in I$. Edges $E = E_{\text{in}} \cup E_{\text{out}}$ are *in-edges* $E_{\text{in}} \subseteq S \times I$ or *out-edges* $E_{\text{out}} \subseteq I \times S$, representing the flow of data during a workflow run. A trace $T$ in this model links atomic data tokens to atomic processes (i.e., having a single process invocation). Data structures are consumed (destroyed) by an invocation via in-edges. Similarly, process invocations create *new* output structures via out-edges. Thus, in this model, processes are viewed as *data transformers*. To avoid write conflicts among multiple invocations, we require that $E_{\text{out}}^{-1} : S \to I$ be a function, associating with each output structure $s \in S$ the unique invocation $i_s \in I$ that created it. A flow graph gives rise to two natural views, a *data dependency graph* $G_d = (S, E_{\text{ddep}})$ and an *invocation dependency graph* $G_i = (I, E_{\text{idep}})$, defined by:

$$E_{\text{ddep}}(s_2, s_1) \longleftarrow E_{\text{in}}(s_1, i),\ E_{\text{out}}(i, s_2)$$
$$E_{\text{idep}}(i_2, i_1) \longleftarrow E_{\text{out}}(i_1, s),\ E_{\text{in}}(s, i_2)$$

Fig. 4a depicts a scientific workflow definition (gray boxes) and Fig. 4b shows a corresponding flow graph in the conventional model. The inferred data and invocation dependency views are shown in Fig. 4c and Fig. 4d, respectively.

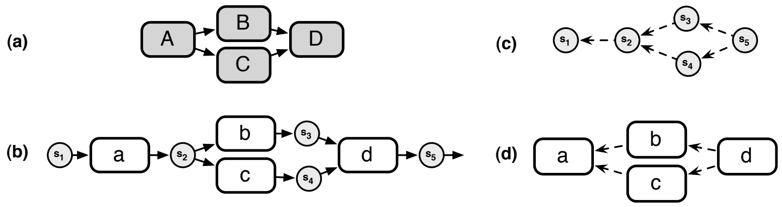


**Fig. 4.** Conventional model: (a) *workflow definition* with steps A, ..., D; (b) example *flow graph* $E = E_{\text{in}} \cup E_{\text{out}}$ with process invocations a,..., d and atomic data $\mathtt{s}_1, \ldots, \mathtt{s}_5$; (c) data dependencies $E_{\text{ddep}}$ and (d) invocation dependencies $E_{\text{idep}}$ inferred from the flow graph $E$ of (b)

**Open Provenance Model.** Our basic model closely ressembles the *Open Provenance Model* (OPM) [18]. In OPM, an atomic data structure $s \in S$ is called an *artifact*, an invocation $i \in I$ is called a *process*, an in-edge $s \rightarrow i$ corresponds to a *used* edge $s \overset{\text{used}}{\dashleftarrow} i$, and an out-edge $i \rightarrow s$ corresponds to a *wasGeneratedBy* edge $i \overset{\text{genBy}}{\dashleftarrow} s$. Similarly, the above dependency views $E_{\text{ddep}}$ and $E_{\text{idep}}$ simply have different names in OPM: For $E_{\text{ddep}}(s_2, s_1)$ we say in OPM that the artifact $s_2$ *was derived from* the artifact $s_1$; and for $E_{\text{idep}}(i_2, i_1)$ we say that the process $i_2$ *was triggered by* the process $i_1$.

**Multi-Invocation Model.** The basic model views processes as *atomic*, i.e., for each task A, B, ... in the workflow definition, there is a *single* invocation a, b, ... in the flow graph (see Fig. 4b). Other models of computation, notably process networks [13] and related dataflow variants with *firing* [14] give rise to finer-grained process models for incremental computations over data streams. Fig. 5a shows the execution of process A modeled as two *independent* invocations, a:1 and a:2, which may be executed concurrently over the input stream $\mathtt{s}_1$, $\mathtt{s}_2$, ... (similarly for B and b:1, b:2). Here, the second invocation a:2 of A does not "see" (is independent of) the earlier input $\mathtt{s}_1$ used by a:1. This is the case, e.g., if A is *stateless*, i.e., has no memory between invocations. Fig. 5b is a variant of Fig. 5a in which A is *stateful*, and thus preserves information between invocations a:$i$, resulting in additional dependencies. More formally, in the *multi-invocation model*, a trace $T = (V, E, \alpha)$ includes a function $\alpha : I \rightarrow A$ returning for each invocation $i \in I$ the *actor* $\alpha(i) \in A$ that created $i$. Conversely, for any actor $A$, $\alpha^{-1}(A) = \{i \in I \mid \alpha(i) = A\}$ is the set of invocations created by $A$ during a workflow run.

The underlying *model of computation* (MoC) of a workflow language determines the kinds of traces that can be generated at execution time. One can understand (and formalize) a MoC as a mapping that associates with a workflow definition $W$ and input $s$, a set of possible traces $T(s)$. The basic model, e.g., with its atomic data tokens and atomic processes can create flow graphs as in Fig. 4b, but not those in Fig. 5, which can support *pipeline parallel* execution.

**Nested Model (Copy Semantics).** So far we have considered data structures as atomic *tokens* $s \in S$, i.e., without further access to any internal structure (e.g., $s$ might denote a string, file, or Java object). This model is often too coarse and thus inadequate when dealing with workflows over nested data such as XML. Thus, we refine the multi-invocation model by "drilling down" to the level of data items within structures. In the nested model with copy semantics, a trace $T = (V, E, \alpha, \tau)$ includes a function $\tau : S \rightarrow X$, which maps structures $s \in S$ to XML trees $\tau(s) \in X$ such that the

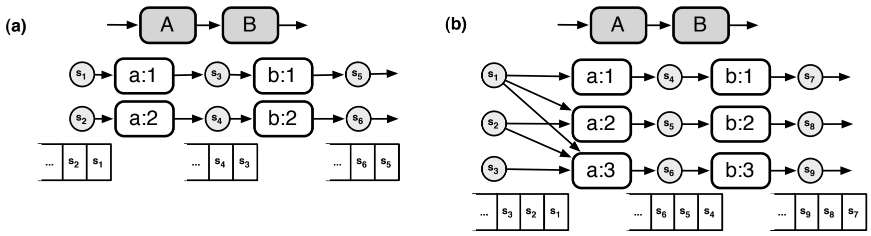


**Fig. 5.** The process network (PN) model supporting *data streaming* and *pipelined execution*: (a) step A of the workflow definition (top) is modeled as two *independent* invocations (a:1, a:2) within the flow graph (bottom), possibly executed concurrently over input stream $s_1, s_2, \ldots$; (b) a variant of (a) where A is *stateful*, preserving information between invocations a:$i$, resulting in additional dependencies (i.e., in edges)

domain $X$ of XML trees is defined as usual. In particular, we assume an underlying space of nodes $N$ from which $X$ is built. As above, we require $E_{\text{out}}^{-1}$ to be a function, i.e., for any $x \in X$ produced, there is a *single* invocation $i_x$ that produced it. This avoids *write conflicts*: no two invocations $i_1$ and $i_2$ can write to the same tree $x$ (like above for $s$). We can think of actors consuming (and destroying) their input and creating "fresh" XML trees for each invocation. Thus, if some data within $x$ must be preserved (e.g., for "pass through"-style processing), a fresh copy $x'$ of $x$ must be created in this model.

**Nested Model (Update Semantics).** Consuming XML objects and recreating parts of them (through fresh copies) can be both inconvenient and inefficient with respect to workflow execution and provenance storage. Under the *update semantics* we assume that different *versions* of trees $\tau(s) \in X$ *share* nodes from $N$ (Fig. 6a).[2] In particular, invocations are modeled as sets of *updates* that produce new versions of their input. Viewing invocations as updates can increase the concurrency of workflow execution for independent invocations. For invocations a:i and a:j of A and inputs $s \in S$, if

$$\varDelta_{\mathtt{a:i}}(\varDelta_{\mathtt{a:j}}(\mathtt{s})) = \varDelta_{\mathtt{a:}j}(\varDelta_{\mathtt{a:}i}(s)),$$

then we say that A has *independent invocations*. In Fig. 6a, A has independent invocations a:1 and a:2, i.e., applying $\varDelta_{\mathtt{a:1}}$ and $\varDelta_{\mathtt{a:2}}$ either in series or in parallel results in $s_2$. There are different ways to achieve this independence. In COMAD, e.g., one can (i) employ non-overlapping scope expressions, and (ii) require further that actors are stateless across multiple invocations of the same actor.

As shown in Fig. 6b, we relax the earlier constraint that $E_{\text{out}}^{-1}$ be a function. Thus, in the nested model with update semantics, a trace $T = (V, E, \alpha, \tau, \gamma)$ includes two functions denoted by $\gamma$, namely, $\gamma+ : N \rightarrow I$ returns the unique invocation that created a node, and $\gamma- : N \rightarrow 2^I$ returns the possibly empty set of invocations that deleted a node.[3] This approach avoids write conflicts at the node level, while still allowing parallel updates to the same tree, e.g., as shown in Fig. 6b. Here we restrict the types of updates that can be performed by invocations to *insertions* and *deletions* of nodes, i.e., similar to COMAD, abitrary structural modifications are not considered.

[2] we can view $s$ as the version-id of $\tau(s)$.

[3] Note that for workflows containing branches, multiple invocations can delete the same node.

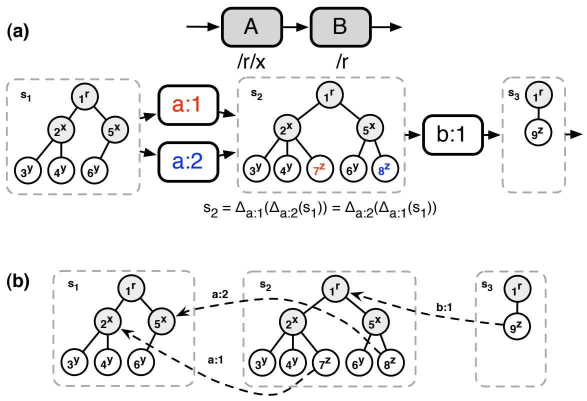


**Fig. 6.** The *unified model*, which supports COMAD and similar models of computation: (a) the complex (i.e., XML) data structure $\mathtt{s}_1$ is *updated* by invocations giving $\mathtt{s}_2 = \Delta_{\mathtt{a:2}}(\Delta_{\mathtt{a:1}}(\mathtt{s}_1))$, and for stateless A, $\mathtt{s}_2 = \Delta_{\mathtt{a:1}}(\Delta_{\mathtt{a:2}}(\mathtt{s}_1))$; and (b) the *labeled dependency (i.e., lineage) edges* representing the correct data dependencies for the nested data structures of (d)

**Unified Model (Fine-Grained Data Dependencies).** Finally, we combine the above models into a unified trace structure supporting both multiple invocations and nested XML with update semantics. Specifically, we extend the previous model with *fine-grained dependencies* for relating nodes according to dependency relationships (as opposed to relating coarse-grained structures as in a flow graph). If a node $n$ is a fine-grained dependency of a node $n'$, then we say that $n$ was directly used in the creation (or derivation) of $n'$. We represent fine-grained dependencies using *lineage relations*, which includes the invocation $i$ that created $n'$, denoted $n \overset{\mathtt{i}}{\dashleftarrow} n'$ in Fig. 6b. Note that if $n'$ depends on a collection node $n$ within the structure $s$, then $n'$ is also assumed to depend on the descendents of $n$ with respect to $s$; and we typically show only the "highest" dependencies of a node, as in Fig. 6b.

A trace in the unified model is of the form $T = (V, E, L, \alpha, \tau, \gamma)$ where fine-grained dependencies are captured by the ternary *lineage relation* $L \subseteq N \times I \times N$. Thus, the unified model consists of three dimensions: (i) *flow relations* among input and output structures (defined by the flow graph), (ii) *parent-child relations* among nodes of structures, and (iii) *lineage relations* defining fine-grained data dependencies. In the following section we consider approaches for querying each dimension separately as well as *hybrid* approaches combining multiple dimensions in a single query.

## 3 Querying Provenance

Our goal is to provide generic support for querying provenance information to enable a wide range of users and applications. Common types of provenance queries we want to support include standard lineage queries [18,6] for determining the data and invocations used to derive other data; queries that allow users to ensure that specific data and invocation dependencies were satisfied within a run; queries for determining the inputs and outputs of invocations (e.g., based on the actors used and their parameters); and queries for determining the structure of data produced by a workflow run. Further, to address shortcomings of current approaches that either do not provide explicit query support or else use query languages not naturally suited for querying provenance, we also seek an approach that satisfies the following desiderata.

**(1) Physical Data Independence.** Many existing approaches for querying provenance information [18] are closely tied to physical data representations, e.g., relational, XML, or RDF schemas, where users express provenance queries using corresponding manipulation languages, i.e., SQL, XQuery, or SPARQL, respectively. Most provenance queries also require computing transitive closures [8,10], and thus expressing provenance queries requires users to have considerable expertise in the underlying schemas and query languages. Some systems also support *multiple* storage technologies (e.g., [2]) in which specific provenance representations are selected by workflow designers. Thus, to query a given workflow run, users must know which representation was used to store the run and be proficient in the associated query language. Instead, a generic approach for querying provenance information should allow multiple representations to be used with the same fundamental query language, and should thus be independent of the underlying physical data model.

**(2) Workflow System Independence.** Many different workflow systems record provenance information. Users should be able to express queries without having to know which workflow system was used to produce provenance information, and without having to use a different language for each system. OPM [17], e.g., is independent of any particular workflow system, whereas the Kepler provenance recorder in [2] requires users to understand the details of Kepler workflow computation models to construct basic lineage information.

**(3) Workflow Definition Independence.** It is often possible to make use of provenance information gathered during a workflow run without accessing or understanding the workflow definition. A generic query approach should allow users to query provenance information without having prior knowledge of the workflow definition, or the types of data input and output by the workflow run. Similarly, a generic query approach should make it possible for users to discover the actors that were invoked during a workflow run, and the types of data used. However, when data types and workflow definitions are known, users should be able to query provenance conveniently via this information.

**(4) Provenance Relationship Preserving.** It is often convenient to visualize provenance information using data and invocation dependency graphs [5,18,8], and these graphs are often constructed from the result of a provenance query. In many provenance approaches (including those based on path expressions [12]), typical queries return only sets of nodes or invocations, requiring additional queries to reconstruct the corresponding lineage graphs. Thus, a general approach for querying provenance information should make it simple for users and applications to construct such graphs by returning paths over lineage information.

**(5) Incremental Queries.** A consequence of preserving provenance relationships within query results is that these results can be further queried. This can allow users to decompose complicated queries into smaller subqueries, and also incrementally refine query results. Thus, a generic query language should be *closed*, i.e., similar to relational algebra, where the result of a provenance query should be queryable using the same language.

**(6) Optimization Transparency.** Because the amount of provenance information produced by a workflow run can be large (e.g., due to the number of workflow steps, and input and output data sets), systems that manage provenance information must provide efficient storage and query approaches [9,8,10,3]. Thus, a generic query approach should be amenable to query optimization techniques, and these optimizations should be independent of the query language itself (i.e., users should not have to modify queries for efficiency).

The first two desiderata are addressed directly through our unified provenance model, which does not depend on any particular workflow system and can be implemented using different underlying provenance storage representations. The rest of this section describes techniques for querying our model, with the goal of address the remaining issues. We first describe how QLP can be used to query the different dimensions of our unified provenance model, and discuss how these dimensions can be combined through *hybrid queries* that mix lineage information with information about the structure of data and the versions of these structures produced by a workflow run. We also give examples of common provenance queries that can be expressed in QLP and discuss our current implementation.

## 3.1 Provenance Queries Using QLP

Fig. 7a shows a portion of the trace for a typical run of the workflow of Fig. 2, which we use below in describing the different types of provenance queries supported in QLP.

**Queries over Nested Data.** Because data structures in our provenance model represent XML trees, these structures can be queried using standard XML languages. In QLP, we adopt XPath as our query language for accessing nested data. For each trace, we also define a simple view that merges the different versions of structures within the trace into a *combined data structure*. As an example, Fig. 7b shows the combined data structure $s$ for the trace of Fig. 7 that consists of the structures $s_1$ to $s_6$. Queries over combined structures provide general access to all the nodes used and produced by a run, e.g., to return nodes of a specific type or with specific metadata annotations. The following XPath expressions are valid QLP queries, which are posed against the combined data structure.

//Image (1)
//AtlasGraphic[modality="speech"]/@* (2)

These queries return (1) the nodes of type Image that were input to or produced by the workflow run, and (2) the metadata annotations (represented as XML attributes) for nodes of type AtlasGraphic with the value "speech" assigned to the (metadata) attribute "modality" [18]. Given a trace $T = (V, E, L, \alpha, \tau, \gamma)$, the combined structure $s$ represents the tree $\tau(s) = \tau(s_1) \cup \cdots \cup \tau(s_n)$ for each $s_i \in V$.[4] XPath queries expressed over combined structures $s$ return ordered subsets of nodes within the tree $\tau(s)$. We note that

[4] Because invocations can only insert and delete nodes, merging two trees is straightforward.

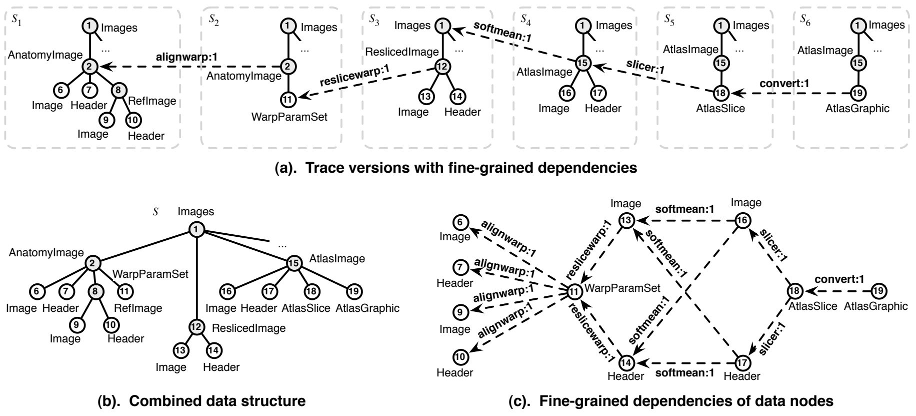


**Fig. 7.** Example trace of a typical run of the workflow in Fig. 2: (a) versions and dependencies created by the first invocation of each actor corresponding to stages 1 through 5; (b) the corresponding combined structure; and (c) the corresponding fine-grained dependency graph

the combined structure of a trace is distinct from the final versions of structures, and instead, contain all data input to and produced by a run (including deleted nodes).

**Queries over Flow Relations (Versions).** In addition to queries expressed over the combined data structure of a trace, QLP also provides constructs for accessing specific versions of structures produced by a workflow run. As mentioned in Section 2, a node in $N$ may occur within multiple versions. We denote an occurrence of a node $n \in N$ within a structure $s$ as $n@s$. The expression $n@s$ *positions* node $n$ at structure $s$ (i.e., $n$ is *positioned* at $s$), whereas the expression $n$ leaves the node *unpositioned*. The following QLP queries use the **@in** and **@out** constructs (see Table 1) to access the different versions of structures within a trace (according to the flow relations).

**@in** (3)
**@out** (4)
**@out** slicer:1 (5)
18 **@out** slicer:1 (6)

These queries return (3) the input structure $s_1$ of the run, (4) the output structure $s_6$ of the run, (5) the version of the output structure $s_5$ produced by the first invocation of the Slicer actor, and (6) the version of node 18 in the structure $s_5$ produced by the first invocation of Slicer.

Unlike querying combined structures, the **@in** and **@out** operators return sets of positioned nodes. For example, the set of nodes returned by query (3) when applied to the example in Fig. 7a would include $1@s_1$, i.e., node 1 positioned at $s_1$ (in addition to all of the descendents of node 1 in $s_1$). The **@in** and **@out** operators return the nodes of structures input to and output by the run when no invocations are given, and when invocations are given, the input and output structures of the invocation, respectively. An (unpositioned) node may also be supplied as an argument to these operators, in which case the set of nodes returned contains only a positioned version of the node (i.e., the node without its descendents), or the empty set if no such node exists in the

**Table 1.** Example QLP constructs and short-hand notations

| *Construct Primitive* | *Shorthand* | *Result* |
|---|---|---|
| $N$ **@in** $I$ | | The version of $N$ input to invocation $I$ (if $I$ not given, at run input) |
| $N$ **@out** $I$ | | The version of $N$ output by invocation $I$ (if $I$ not given, at run output) |
| $N_1$ **derived** $N_2$ | $N_1..N_2$ | Lineage edges forming *transitive* paths from $N_1$ to $N_2$ |
| $N_1$ **1_derived** $N_2$ | $N_1.N_2$ | Lineage edges forming *one-step* paths from $N_1$ to $N_2$ |
| $N_1$ **through** $I$ **derived** $N_2$ | $N_1..\#I..N_2$ | Lineage edges forming *transitive* paths from $N_1$ to $N_2$ through invocation $I$ |
| $N_1$ **through** $I$ **1_derived** $N_2$ | $N_1.\#I.N_2$ | Lineage edges forming *one-step* paths from $N_1$ to $N_2$ through invocation $I$ |
| **type** $N$ | | The type (tag name) of a node $N$ |
| **actors** $L$ | | Actors of invocations in lineage edges $L$ |
| **invocations** $L \cup A$ | | Invocations of lineage edges $L$ or of an actor $A$ |
| **nodes** $L$ | | Nodes of lineage edges $L$ |
| **input** $L$ | | Source nodes of lineage edges $L$ |
| **output** $L$ | | Sink nodes of lineage edges $L$ |

corresponding structures. For example, query (6) applied to the run graph of Fig. 7a returns the positioned node $18@s_5$.

**Queries over Lineage (Fine-Grained Dependencies).** The majority of provenance queries are expressed over lineage relations (e.g., see [18,6]). Fig. 7c is an example of a portion of the lineage relations for the trace of Fig. 7a, showing only the fine-grained dependencies of data nodes (i.e., collection nodes are not shown). The QLP operators for querying lineage (see Table 1) act as filters over lineage relations $L \subseteq N \times I \times N$, returning subsets of $L$. Thus, in addition to query results maintaining lineage relations, these results can be further queried via QLP lineage operators. For example, consider the following simple QLP lineage queries.

$$* \textbf{ derived } 19 \tag{7}$$
$$6 \textbf{ derived } * \tag{8}$$
$$* \textbf{ through } \text{slicer:1} \textbf{ derived } * \tag{9}$$

These queries return (7) lineage relations denoting the set of paths starting at any node and ending at node 19, (8) lineage relations denoting the set of paths starting at node 6 and ending at any node, and (9) lineage relations denoting the set of paths with any start and end node, but that go through the first invocation of the Slicer actor.

Given a set of lineage relations $L$, let $paths(L)$ be the set of paths implied by $L$, where a lineage path takes the form

$$n_1.i_1.n_2 \ldots i_j \ldots n_{k-1}.i_{k-1}.n_k \qquad (k \geq 2)$$

For any set $P$ of such paths, let $edges(P)$ be the set of lineage relations for $P$, such that $L = edges(paths(L))$. Given lineage relations $L$, the query '$n_1$ **derived** $n_k$' returns the subset of $L$ consisting of paths of any length starting at $n_1$ and ending at $n_k$. Similarly, the query '$n_1$ **through** $i_j$ **derived** $n_k$' returns the subset of $L$ consisting of paths of any length starting at $n_1$ and ending at $n_k$ that also pass through invocation $i_j$. If **1_derived** (i.e., one-step or immediately derived) is used in place of **derived**, then only paths of the form $n_1.i_j.n_k$ are returned. That is, in **1_derived** each selected path is defined by a single lineage relation. These operators can be combined to form more complex lineage paths. In this case, each individual expression specifies a segment of a larger lineage path. For instance, a query such as '$n_1$ **derived** $n_j$ **derived** $n_k$' selects lineage relations that form paths from $n_1$ to $n_j$ and from $n_j$ to $n_k$.

Lineage queries may also be expressed using a shorthand notation (see Table 1). Similar to the use of '/' and '//' in XPath expressions for navigating parent-child relationships of nodes, we use '.' to navigate immediate (i.e., one-step) lineage paths and '..' to navigate transitive lineage paths (i.e., paths consisting of one or more steps). When using the shorthand notation, we distinguish actor invocations from structural nodes by prefixing invocations with '#'. For example, queries (7-9) can be expressed as '*..19', '6..*', and '*..#slicer:1..*' (or simply '#slicer:1') using the QLP shorthand notation.

**Hybrid Queries over Multiple Dimensions.** Lineage queries can be combined with queries over data structures. We call these "hybrid" queries since they allow both structural and lineage information to be accessed simultaneously. The following are simple examples of QLP hybrid queries.

* **derived** //AtlasImage//* (10)
//ReslicedImage//* **through** softmean:1 **derived** //AtlasImage (11)
//Image **through** Slicer[x="0.5"] **derived** //AtlasImage (12)
(//* @**in** slicer:1) **derived** //AtlasImage (13)

These queries return lineage relations denoting (10) paths ending at descendents of AtlasImage nodes, (11) paths starting at ReslicedImage descendent nodes and ending at AtlasImage nodes that pass through the first invocation of SoftMean, (12) paths starting at Image nodes and ending at AtlasImage nodes that pass through invocations of Slicer with parameter "x" set to the value 0.5 (i.e., resulting in "Atlas X Images"), and (13) paths starting at (positioned) input nodes of the first invocation of Slicer and ending at AtlasImage nodes. By returning all lineage relations, the result of these queries can be easily visualized as lineage graphs and can be treated as views that can be further queried using similar expressions.

Hybrid queries can be (naively) evaluated by (i) obtaining the structures resulting from @**in** and @**out** version operators, (ii) applying XPath expressions to these structures, and (iii) applying lineage queries to the resulting nodes. For example, when applied to Fig. 7, query (10) is first evaluated by executing the XPath query '//AtlasImage//*' over the combined structure *s*, returning nodes 16–19. For each node, a separate lineage query is evaluated, i.e., '* **derived** 16', '* **derived** 17', '* **derived** 18', and '* **derived** 19', such that the answer to query (10) is the unique set of resulting lineage relations.

Hybrid queries may also combine lineage operators, e.g., consider the following query that returns the lineage relations denoting paths through SoftMean, Slicer, and Convert invocations, and ending at nodes of type AtlasGraphic.

* **through** Softmean **derived** * **through** Slicer **derived** * **through** Convert (14)
**derived** //AtlasGraphic

For queries that specify complex lineage paths, it is often more convenient to use the shorthand notation, e.g., query (14) can be equivalently written as

#Softmean .. #Slicer .. #Convert .. //AtlasGraphic (short-hand version of 14)

QLP also supports additional operations that can be applied to sets of lineage relations, which are summarized in Table 1 and used in the following examples.

**invocations**(AlignWarp[m="12", dateOfExecution="Monday"]) (15)

**output**(//Header[max="4096"] **derived** //AtlasGraphic) (16)
**output**(#AlignWarp[m="12"] .. #Softmean) (17)
**input**(//Image **derived** //AtlasGraphic @**out**) (18)
**actors**(//Image **derived** //AtlasGraphic @**out**) (19)
(//Image @**in**) − **input**(//Image @**in derived** //AtlasGraphic @**out**) (20)

Query (15) returns the set of invocations of AlignWarp that used the values 12 and "Monday" for the "m" and "dateOfExecution" parameters, respectively. Query (16) returns the output nodes for lineage paths starting at Header nodes having the value "4096" for the global maximum metadata field "max" and ending at AtlasGraphic nodes. Specifically, the **output** operation (similarly, **input**) returns the nodes within lineage relations that do not have outgoing (incoming) lineage edges. Query (17) returns the output nodes produced by invocations of SoftMean that were derived from AlignWarp invocations using the value 12 for parameter "m". This query combines multiple lineage operators and is expressed using the QLP shorthand notation. Query (18) returns the input nodes of paths starting from Image nodes and ending at AtlasGraphic nodes that were part of the output of the run. Query (19) is similar to (18) but returns the actors of invocations used in the corresponding lineage paths. Query (20) finds the Image nodes input to the workflow run that were not used to derive any output Image nodes. This is achieved by first finding all Image nodes input to the workflow run, and then subtracting from this set the input Image nodes used to derive an output AtlasGraphic node. Although not shown here, the **type** operator of Table 1 can be used to return the tag names of XML nodes.

QLP also provides support for constraining the structure of lineage paths using regular expressions. For instance, the query '$n_1$ **through** ($i_1$ | $i_2$) **derived** $n_2$' selects lineage relations denoting paths from $n_1$ to $n_2$ that pass through either $i_1$ or $i_2$. These queries can be used by workflow developers, e.g., to ensure that complex workflows (e.g., involving multiple branches) executed correctly.

## 3.2 Implementation and Evaluation of QLP Query Support

Our current implementation of QLP supports queries expressed using the shorthand notation described in Table 1, and answers these queries using the relational storage strategies described in [3]. In particular, we have implemented a Java application that takes as input a QLP query, transforms the query into an equivalent SQL query, and executes the SQL query over the underlying database. Using our implementation, we were able to express and answer the queries of the first provenance challenge [18] as well as queries similar to those in [6]. A number of variants of these queries are also given in (1–20) above. The QLP versions of these queries are significantly more concise than those typically expressed against underlying storage structures [18], and in general, are easier to formulate and comprehend.

The storage strategies described in [3] apply reduduction techniques for efficiently storing both immediate and transitive provenance dependencies. In particular, a naive approach for storing nodes $N$ and their dependencies $D$ is as tuples $P(N, D)$ in which nodes involved in shared dependencies will be stored multiple times. For example, if nodes $n_4$, $n_5$, and $n_6$ each depend on nodes $n_1$, $n_2$, and $n_3$, nine tuples must be stored

$P(n_4,n_1)$, $P(n_4,n_2)$, $P(n_4,n_3)$, $P(n_5,n_1)$, ..., $P(n_6,n_3)$, where each node is stored multiple times in $P$. Instead, the approach in [3] introduces additional levels of indirection through "pointers" (similar to vertical partitioning) for storing reduced sets of dependencies. Thus, we divide $P(N,D)$ into two relations $P_1(N,X)$ and $P_2(X,D)$ where $X$ denotes a pointer to the set of dependencies $D$ of $N$. For instance, using this approach we store only six tuples $P_1(n_4,\&x)$, $P_1(n_5,\&x)$, $P_1(n_6,\&x)$, $P_2(\&x,n_1)$, $P_2(\&x,n_2)$, and $P_2(\&x,n_3)$ for the above example. Additional levels of indirection are also used to further reduce redundancies within dependency sets based on their common subsets, and similar techniques are used to reduce transitive dependency sets by applying reduction techniques directly to pointers (as described in [3]).

As an initial evaluation of the feasibility and scalability of our approach for executing QLP queries, we describe below the results of executing lineage queries over synthetic traces of increasing numbers of nodes and lineage relations. We compare our *reduced-transitive approach* (R), which transforms QLP queries to SQL queries over our underlying relational schema storing immediate and transitive lineage relations in reduced form, to a *naive approach* (N) in which only immediate dependencies are stored, and corresponding QLP lineage queries are transformed to recursive stored procedures.

Our experiments were performed using a 2.4GHz Intel Core 2 duo PC with 2 GB RAM and 120 GB of disk space. Each approach used MySQL to store provenance information. We compare query response time and storage size using synthetic traces ranging from 100 to 3000 nodes, $10^3$ to $10^4$ immediate dependencies, $10^4$ to $10^6$ transitive dependencies, and lineage paths of length 25 to 150, respectively. The synthetic traces were taken from [3], and represent typical lineage patterns generated by real-world workflows implementing phylogenetic and image-processing pipelines [3,5,18].

We consider the following basic lineage queries for evaluating query response time. In general, queries over lineage relations are considerably more expensive [10,3] than queries over only combined structures (i.e., XPath queries), or queries that select only versions of structures within a trace (e.g., using the **@in** and **@out** QLP operators).

$$* \mathinner{..} n \tag{Q1}$$
$$n \mathinner{..} * \tag{Q2}$$
$$\textbf{exists}\ n_1 \mathinner{..} n_2 \tag{Q3}$$
$$n_1 \mathinner{..} n_2 \tag{Q4}$$
$$n_1 \mathinner{..} n_2 \mathinner{..} n_3 \tag{Q5}$$

These queries return (Q1) lineage relations denoting paths that lead to a node $n$ (e.g., to return the full lineage of $n$); (Q2) lineage relations denoting paths that start from a node $n$ (i.e., the "progeny" of $n$ [6]); (Q3) true if there is a lineage path from node $n_1$ to node $n_2$ (where '**exists**' denotes a boolean query); (Q4) lineage relations denoting paths starting at node $n_1$ and ending at node $n_2$; and (Q5) lineage relations denoting paths starting at node $n_1$ and ending at node $n_3$ that pass through node $n_2$.

The left side of Fig. 8 shows that the query response time for the reduced-transitive implementation grows linearly with increasing trace size, and is significantly faster than the naive approach (shown in dotted lines). The more expensive query response time of the naive approach is due to the use of recursive stored procedures to select transitive dependency relationships. Moreover, the increased query response time for the reduced-transitive approach, when compared to the naive approach, is not simply based

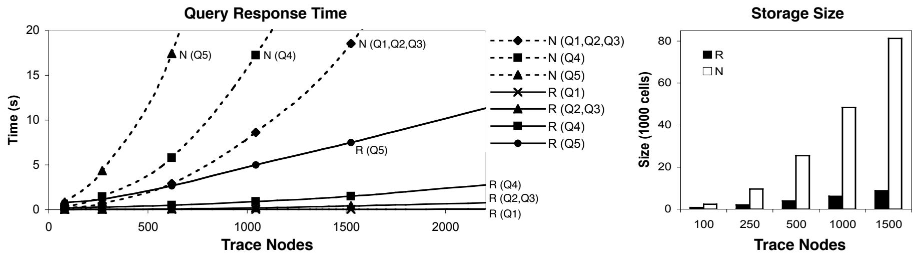


**Fig. 8.** Query response time (left) and storage size (right) for the reduced-transitive and naive approaches. In the reduced-transitive approach, because of the nature of the reduction strategies and the QLP-to-SQL translation, query Q1 takes (slightly) less time than query Q2, and similarly, query Q2 takes the same amount of time as query Q3 for the example traces.

on increases in storage size. In particular, the right side of Fig. 8 shows that when compared to the naive approach, storage size is also significantly smaller using our reduction strategies. Thus, we can improve query response time by materializing transitive lineage relations, while at the same time reducing the provenance storage size using reduction techniques. Fig. 8 also demonstrates that the standard approach of using recursive stored procedures to answer lineage queries does not scale linearly with trace size.

These results highlight the advantages of decoupling provenance query languages from underlying storage schemes. In particular, the same QLP queries can be answered using different storage strategies, in this case using the naive and reduced-transitive approaches, to transparently improve query response time and provenance storage size.

## 4 Related Work

Conventional models of provenance [17,8,2,9] are largely based on the assumption that data structures and processes are atomic. These assumptions do not hold, however, for many computation models employed within current scientific workflow systems [16,15]. Our approach extends the conventional model by supporting workflows composed of actors that can have multiple invocations (e.g., as in process networks [15]) and that employ update semantics over complex data structures (in particular, XML). This paper extends our prior work [3] on efficiently storing provenance information by defining a logical representation of the model and corresponding query approaches (i.e., QLP).

Few query languages have been specifically developed for querying provenance information. Instead, most approaches (e.g., see [18,22]) define their provenance models in terms of physical schemas represented within general-purpose data management frameworks (e.g., for storing relational, XML, or RDF data), and provenance queries are expressed using the corersponding query languages of the framework (e.g., SQL, XQuery, or SPARQL). This approach often leads to complex query expressions, even for answering basic provenance questions. For instance, to query over lineage information stored within a relational database, users must typically specify join operations over multiple tables and compute transitive closures [9,8,10,3]. Our approach differs by separating the logical provenance model from its physical representation and providing provenance-specific query constructs. Thus, users express queries against the logical model using more natural constructs, which are automatically translated to equivalent queries expressed against the underlying physical representation.

Similar to our approach, the VisTrails provenance query language (vtPQL) [21] defines provenance-specific query constructs. However, vtPQL only supports the standard provenance model, provides a limited set of constructs for acessing provenance relations, and does not support queries over data structures. For example, vtPQL defines *upstream*($x$) to return all modules (i.e., actors) that procede $x$ in the workflow definition. The upstream operator is the primary construct for answering lineage queries related to invocations, where a query "upstream(x) − upstream(y)" is used to find dependencies between two invocations of modules $x$ and $y$ within a run. However, vtPQL assumes only a *single* lineage path between two such modules, and thus would return incorrect results in the case of multiple lineage paths. Further, queries related to exclusive data lineage (e.g., $n_1$ .. $n_2$, $n_1$ .. $n_2$ .. $n_3$, $n_1$ .. #$a$ .. $n_2$) are not supported in vtPQL.

In [12], Lorel [1] is used to represent and query provenance information. Similar to our approach, [12] employs generalized path expressions for querying lineage information. While both vtPQL and our approach provide a closed language over lineage edges, e.g., to support the visual representation of provenance lineage graphs, languages based on path expressions (Lorel, XPath, OQL, etc.) primarily return sets of identifiers (e.g., representing nodes and invocations) that require additional queries for constructing lineage graphs. Further, approaches such as [12] are still closely tied to physical representations of provenance.

## 5 Conclusion

We have described a logical model of provenance for representing scientific workflow runs based on computation models that work over XML structures. Our approach naturally extends the conventional provenance model by adding support for nested data and for accurately capturing detailed lineage information of processes employing update semantics. We also described a general approach for querying provenance using our Query Language for Provenance (QLP), which provides specialized constructs for expressing both structural (i.e., parent-child) and lineage (i.e., data dependency) "hybrid" queries. We have also shown how these constructs can be used to express a wide range of provenance queries (including those of [18]) and that answering QLP queries using relational technology can be feasible when both immediate and transitive dependency edges are stored according to [3]. As future work we plan to extend our current implementation of QLP with additional optimization techniques, with the goal of providing a generic and efficient approach for addressing challenges in managing the provenance of scientific workflow runs.

**Acknowledgements.** This work supported in part through NSF grants IIS-0630033, OCI-0722079, IIS-0612326, DBI-0533368, ATM-0619139, and DOE grant DE-FC02-01ER25486.

## References

1. Abiteboul, S., Quass, D., McHugh, J., Widom, J., Wiener, J.L.: The Lorel Query Language for Semistructured Data. Intl. J. on Digital Libraries (1997)
2. Altintas, I., Barney, O., Jaeger-Frank, E.: Provenance Collection Support in the Kepler Scientific Workflow System. In: Moreau, L., Foster, I. (eds.) IPAW 2006. LNCS, vol. 4145, pp. 118–132. Springer, Heidelberg (2006)
3. Anand, M.K., Bowers, S., McPhillips, T., Ludäscher, B.: Efficient Provenance Storage Over Nested Data Collections. In: EDBT (2009)
4. Buneman, P., Suciu, D.: IEEE Data Engineering Bulletin. Special Issue on Data Provenance 30(4) (2007)
5. Bowers, S., McPhillips, T., Riddle, S., Anand, M., Ludäscher, B.: Kepler/pPOD: Scientific Workflow and Provenance Support for Assembling the Tree of Life. In: Freire, J., Koop, D., Moreau, L. (eds.) IPAW 2008. LNCS, vol. 5272, pp. 70–77. Springer, Heidelberg (2008)
6. Bowers, S., McPhillips, T., Ludäscher, B., Cohen, S., Davidson, S.B.: A Model for User-Oriented Data Provenance in Pipelined Scientific Workflows. In: Moreau, L., Foster, I. (eds.) IPAW 2006. LNCS, vol. 4145, pp. 133–147. Springer, Heidelberg (2006)
7. Callahan, S., Freire, J., Santos, E., Scheidegger, D., Silva, C., Vo, H.: VisTrails: Visualization Meets Data Management. In: SIGMOD (2006)
8. Chapman, S., Jagadish, H.V., Ramanan, P.: Efficient Provenance Storage. In: SIGMOD (2008)
9. Davidson, S.B., Freire, J.: Provenance and Scientific Workflows: Challenges and Opportunities. In: SIGMOD (2008)
10. Heinis, T., Alonso, G.: Efficient Lineage Tracking for Scientific Workflows. In: SIGMOD (2008)
11. Hidders, J., Kwasnikowska, N., Sroka, J., Tyszkiewicz, J., den Bussche, J.V.: Petri Net + Nested Relational Calculus = Dataflow. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3760, pp. 220–237. Springer, Heidelberg (2005)
12. Holland, D., Braun, U., Maclean, D., Muniswamy-Reddy, K.K., Seltzer, M.: A Data Model and Query Language Suitable for Provenance. In: IPAW 2008 (2008)
13. Kahn, G.: The Semantics of a Simple Language for Parallel Programming. In: IFIP Congress, vol. 74 (1974)
14. Lee, E.A., Matsikoudis, E.: The Semantics of Dataflow with Firing. In: From Semantics to Computer Science: Essays in memory of Gilles Kahn. Cambridge University Press, Cambridge (2008)
15. Ludäscher, B., et al.: Scientific Workflow Management and the Kepler System. Conc. Comput.: Pract. Exper. 18(10) (2006)
16. McPhillips, T., Bowers, S., Zinn, D., Ludäscher, B.: Scientific Workflow Design for Mere Mortals. Future Generation Computer Systems 25(5) (2009)
17. Moreau, L., Freire, J., Futrelle, J., McGrath, R., Myers, J., Paulson, P.: The Open Provenance Model. Tech. Rep. 14979, ECS, Univ. of Southampton (2007)
18. Moreau, L., et al.: The First Provenance Challenge. Conc. Comput.: Pract. Exper., Special Issue on the First Provenance Challenge 20(5) (2008)
19. Oinn, T., et al.: Taverna: Lessons in Creating a Workflow Environment for the Life Sciences. Conc. Comput.: Pract. Exper. 18(10) (2006)
20. Qin, J., Fahringer, T.: Advanced Data Flow Support for Scientific Grid Workflow Applications. In: ACM/IEEE Conf. on Supercomputing (2007)
21. Scheidegger, C., Koop, D., Santos, E., Vo, H., Callahan, S., Freire, J., Silva, C.: Tackling the Provenance Challenge One Layer at a Time. Conc. Comput.: Pract. Exper. 20(5) (2008)
22. Simmhan, Y.L., Plale, B., Gannon, D.: A survey of data provenance in e-science. SIGMOD Record 34(3) (2005)

# Data Integration with the DaltOn Framework – A Case Study

Stefan Jablonski[1], Bernhard Volz[1], M. Abdul Rehman[1], Oliver Archner[1], and Olivier Curé[2]

[1] University of Bayreuth, Germany
{stefan.jablonski,bernhard.volz,abdul.rehman}@uni-bayreuth.de, oliver.archner@bayceer.uni-bayreuth.de
[2] Université Paris-Est, IGM, France
olivier.cure@univ-paris-est.fr

**Abstract.** Data integration has gained a great attention in current scientific applications due to the increasingly high volume of heterogeneous data and proliferation of diverse data generating devices such as sensors. Recently evolved workflow systems contributed a lot towards scientific data integration by exploiting ontologies. Even though they offer good means for modeling computational workflows, they were proved not to be sufficiently strong in addressing data related issues in a transparent and structured manner. The DaltOn system improves the productivity of scientists by providing a framework which copes with these issues in a transparent and well structured manner. In this paper we will elaborate its application in a real world scenario taken from meteorological research where data are retrieved from a sensor network and are integrated into a central scientific database.

## 1 Introduction

The use of sensor networks has recently evolved in many scientific fields like environmental and ecological sciences. These sensors generate streams of raw data at temporal and spatial granularities. Before being used in any analytical experiment, these data are required to undergo several processing steps of integration and transformation since data, stemming from different sensors, are highly heterogeneous in terms of format, syntax, structure and semantics. Besides these data heterogeneity issues, proper transportation of data between sources and sinks, data validation, data filtering and imposing data constraints are also main challenges. In order to tackle them, workflow technology played a vital role in scientific domains. In recent years, scientific workflow systems (e.g. Kepler [7], Taverna [3]) appeared. These systems offer good means to model and execute computational scientific workflows, but they are lacking in addressing data management issues in transparent manner.

A single scientific workflow is split into two categories of work steps. First, those which are application specific for instance applying an 'R' algorithm. Second, the work steps which are related to data operations such as data validation, filtering, integration and transformation. A framework that implements this approach is DaltOn (Data Logistics with Ontologies). DaltOn is meant to be used inside scientific workflow systems for handling data transfer and transformations transparently such that the scientific users can solely concentrate on the scientific analysis while a data experts concentrate on the integration task by means of DaltOn. The method which combines DaltOn with the Perspective Oriented Process Modeling approach is described in [5], its architecture and semantic integration algorithms in [6]. We now want to present a comprehensive application of the DaltOn integration framework in meteorological research.

## 2 Motivating Scenario

Meteorological studies try to increase the understanding of the atmosphere by merging information delivered by diverse sensors. For example to figure out the role of fog deposition in biogeochemical cycles scientists need to observe the fog using a *Present Weather Detector* (PWD) in combination with a *Ultrasonic Anemometer* (UA) for measuring the wind and a chemical analysis of condensed fog water [11].

The Data Group of the Bayreuth Center of Ecology and Environmental Research (BayCEER) supports meteorologists with a data acquisition infrastructure composed of a sensor network consisting of nearly 150 sensors, some import procedures (e.g. interpolation of data for cleansing operations) and a database backend. The system was built up in 1980 and has been continuously improved since then. Fig. 1 shows a portion of the workflow enacted before applying the DaltOn integration framework.

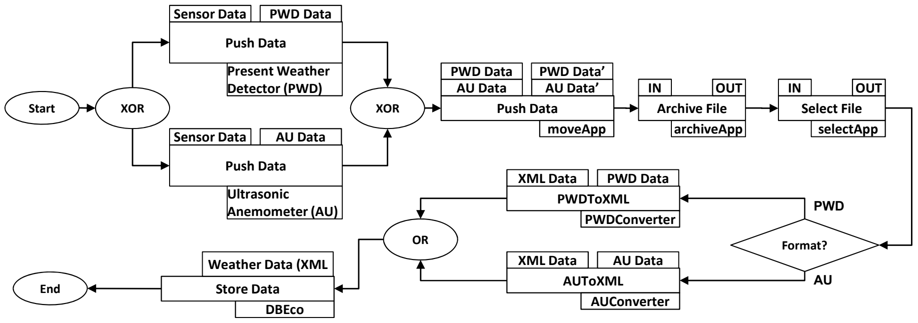


**Fig. 1.** Workflow (without DaltOn) showing the data flow from sensors to databases

In a first step, data are pushed by sensors in the PWD or UA formats over a serial communication line into a file on a PC. The format of these files is vendor specific. Fig. 2 shows a sample output file for a PWD sensor as a CSV formatted file. A unique feature of this device is its capability to estimate the weather type like fog, snow or rain according to NWS [9] and WMO [10] code as shown by columns 7 to 10 in Fig. 2. A system job then uploads the file to a central server. In the following step all files are archived on a backup file server for later reference. After that, each file is selected and converted from its proprietary format into an XML representation which is used by the database import step. Conversions are performed by custom procedures implemented in Perl and Java.

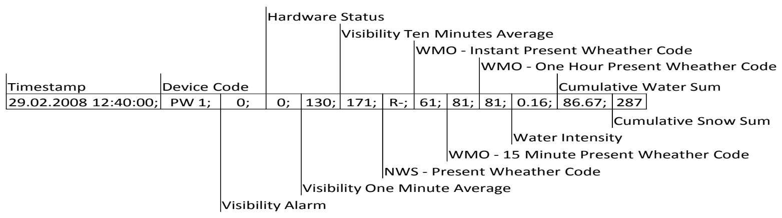

Hardware Status
Visibility Ten Minutes Average
WMO - Instant Present Wheather Code
WMO - One Hour Present Wheather Code
Timestamp
Device Code
Cumulative Water Sum
29.02.2008 12:40:00; PW 1; 0; 0; 130; 171; R-; 61; 81; 81; 0.16; 86.67; 287
Cumulative Snow Sum
Water Intensity
WMO - 15 Minute Present Wheather Code
NWS - Present Wheather Code
Visibility One Minute Average
Visibility Alarm

**Fig. 2.** Example contents of a PWD file

The sensor data is combined with older data in the database to form a time series; since the sensors have no knowledge about these time series and how it is constructed, additional mappings are required for the integration system. The database import step then terminates the sample workflow.

## 2.1 Identified Issues in the Current Implementation

We now explain some of the main issues that we identified in the current implementation of the scenario introduced above. One issue is the 'incomplete transport of information' from sensors to the data store. This is caused by the custom transformation programs which rely on configuration files and that do not check whether certain assumptions made about data are true or not. For instance it is possible that the position of values in the CSV document differ due to a reconfiguration of a sensor. Then this can yield wrong values in a time series. Another issue is the 'lack of data validation and filtering'. Typical problems are truncated files caused by interrupted transfers which are rather often occurring due to a high number of electrical power outages on the sites. In some cases, sensor devices also produce out of benchmark values due to hardware problems. Then these values must be filtered out. The current implementation requires a manual intervention for detecting and deleting the corrupted files and invalid values. Another issue is the 'management of diverse data formats' for the configuration files; scientists must carefully deal with them in order to identify the right column number and to edit the configuration files properly. During the life-span of the installation it is very likely that a device is replaced by a newer one or an existing device needs an alteration of its configuration. To handle these kinds of events meta-information about the device must be updated, new converter must be developed and configuration files must be extended. Right now all these tasks are performed manually by several employees.

## 3 The DaltOn Integration Framework

DaltOn is a framework that efficiently deals with data management issues such as heterogeneity of formats and syntactic as well as semantic incompatibilities. For modeling scientific workflows we use POPM approach [4] since its separation of concerns in different perspectives (Functional, Behavioral, Data and Data Flow, Organizational and Operational) greatly fosters our approach. The purpose of the data and data flow perspective is to describe where data is produced in a process, where it is consumed, and what schema and ontologies it references. The data perspective is converted into single *Data Logistics Workflows* (DaLo-WFs) during the enactment of a POPM process. A DaLo-WF is again a normal workflow but that solely describes the transportation and transformation of data from a source to a corresponding sink in terms of operations provided by DaltOn. A data source and a sink are defined by the applications that produce and consume a certain data item.

In order to perform such an integration task successfully, DaltOn requires some prerequisite steps to be performed by the user during set-up of the task. These are:

- *Semantic description of data sources*: Each data source (and sink respectively) involved in an integration task is described semantically by a so-called *local ontology*. The local ontology is constructed out of a vocabulary which is global with respect to the application domain, the specific application or one specific DaLo-WF and is referred to as *global* or *reference ontology*. DaltOn adopts Description Logics (DL, [1]) for representing all ontologies.
- *Construction of global or reference ontology/ies* - DaltOn exploits multiple ontologies. As described above the local ontologies are used for describing data sources semantically and are thus only valid for one specific source. Nevertheless the two local ontologies (one for the data source and one for the data sink) involved in a DaLo-WF must share one vocabulary. This vocabulary is collected and specified in the global or reference ontology. Much effort has been put into the construction of such global ontologies for complete application domains ([2], [8]). Nevertheless cases can occur in which data is to be transferred in between steps of interdisciplinary nature. Then a reference ontology can be constructed that connects the two domains for one specific case. However this reference ontology is only valid for that specific DaLo-WF. The concept of the global or reference ontologies makes local ontologies comparable and allows for finding and performing matches in between concepts of the local ontologies.

## 3.1 Architecture of DaltOn

DaltOn consists of various logical components [6] as shown in Fig. 3. These logical components are further categorized by three conceptual abstractions namely *Data Provision*, *Data Operation* and *Data Integration*.
*Data Provision* offers the functionality of physical data exchange between data sources – in the context of a process, between a data producing step (source) and a data consuming step (sink). The components under the aegis of this abstraction are:

- *Data Transportation*: manages the data movement between data sources and DaltOn, e.g. FTP transfers. It has no knowledge about the data itself (syntax, structure, semantic) nor does it know what portion of data are to be extracted in case of large data volumes, nor does it know how to insert data into a specific sink of a transportation task.
- *Data Extraction / Selection*: This component constitutes the library of functions which are useful for extraction/insertion and selection of data.

*Data Operation* aims at introducing two functionalities into the system. Besides taking care of assorted data formats, it also offers the mechanism to implement custom functions, e.g. arithmetic calculations, on data values. The components contained in this abstraction are:

- *Format Conversion*: A library of functions that converts source specific formats into an XML representation and from there back to the specific sink format.
- *Data Preparation*: This component plays a special role within DaltOn since it is the only one that actually manipulates data values. Its function library comprises for instance methods for applying interpolations, performing arbitrary arithmetic calculations, or applying aggregation functions on a dataset.

*Data Integration* is the most important abstraction of DaltOn, aiming at integrating data semantically as well as structurally. It also consists of two components:

- *Semantic Integration:* This component implements the functions that are purely related to the detection and resolution of semantic conflicts. It also provides a solution for a terminological transformation; for instance in our use case sensors are identified by a single field called *devicecode* in the extracted dataset but in the database by two fields called *devicename* and *deviceID*.
- *Structural Integration:* This component provides the capability of integrating datasets based on their structures. It assumes that there is no semantic conflict between source and sink datasets, but datasets are incompatible in terms of their structures. For instance it may include the functions for merging, splitting, and concatenating datasets / data records.

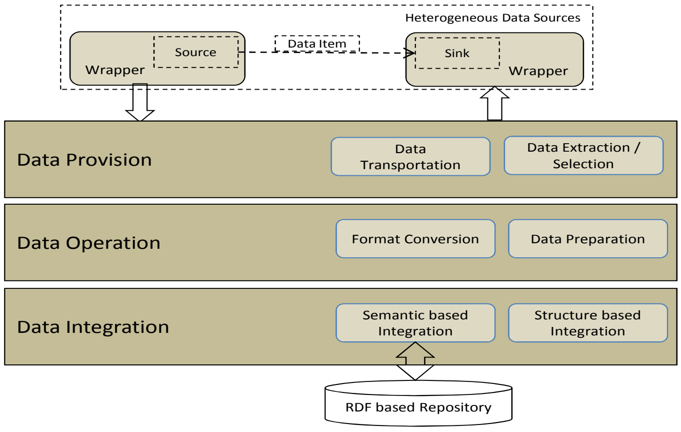


**Fig. 3.** Conceptual abstractions of DaltOn

## 4 Data Integration with DaltOn

Before detailed discussion on how DaltOn performs data integration, we would like to demonstrate how our system facilitates a normal domain user on an abstract level. The workflow presented in Fig. 1 shows the overall use case scenario modeled in a conventional way without DatltOn. The tasks of the process shown in the model can be separated into two categories. One, *Application Specific Tasks*: Tasks which are used to perform application related operations, e.g. scientific analysis. Work steps such as *Push Data*, *Archive File* and *Store Data* in Fig. 1 may belong to this category. We call the workflow containing work steps of this category the Application

Workflow (AWF). Second, *Data Specific Tasks*: Tasks which are purely related to data operations and normally of no interest for a domain user. For instance work steps *PWDToXML* and *UAToXML* of Fig. 1 belong to this category. The workflow containing this type of work steps is called Data Logistic Workflow (DaLo-WF). AWFs are specified by domain users (for instance scientists) on a very abstract level whereas a DaLo-WF between every two work steps of an AWF is automatically generated by DaltOn system. Thus with DaltOn the data specific tasks do not need to be modeled explicitly; instead all these tasks are performed by DaltOn implicitly based on the data flow specified within the process. Fig. 4 depicts the AWF of workflow shown in Fig. 1. The AWF – which is visible to the domain user – is much easier to comprehend since it hides those steps which are purely related to data integration.

In order to discuss the data integration operations performed by DaltOn, we zoom into one portion of the process and focus on two work steps namely *Select File* and *Store Data.* The step *Select File* selects an arbitrary file with sensor data which can be given in the PWD or UA format (here we assume a PWD dataset). This file is then transmitted to the step *Store Data* which is responsible for populating data into the database of the institution. Whenever data exchange between two work steps of AWF is desirable, DaltOn can generate and parameterize a DaLo-WF automatically. This generation and parameterization is driven by the configuration data provided by a data expert at the time of modeling input and output ports of involved work steps in the AWF. A discussion on how a DaLo-WF is derived is out of the scope of this paper. Fig. 5 presents the generated DaLo-WF between *Select File* and *Store Data* work steps.

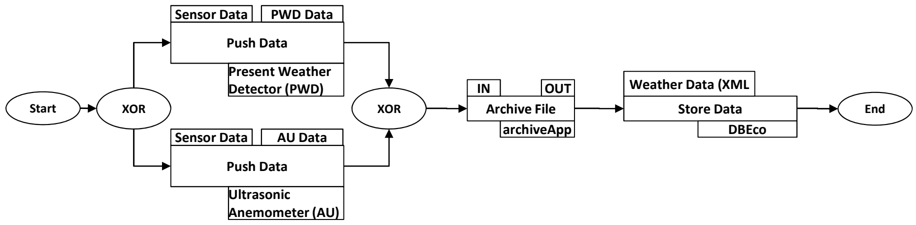


**Fig. 4.** Workflow (with DaltOn) showing the data flow from sensors to databases

A detailed description of each work step of the DaLo-WF depicted in Fig. 5 is given in the following:

- *Data Extraction*: In this step datasets are retrieved from the source. DaltOn component 'DES' is exploited to perform this task and is provided with some configuration data as parameters. 'Extraction Criteria' identifies the actual dataset to be selected, for instance a file name 'pwd_1615.pwd' in the example scenario. 'Source Credential' is the login information for the data source. 'Source Ref' indicates the location of the data source. In order to actually retrieve the data this component determines which wrapper is responsible for the data source. Fig. 6a shows extract of 'PWD' data after the retrieving from file server.
- *Data Transportation*: In this step DT component is utilized that physically moves the extracted datasets to the data staging area of DaltOn for further processing. It requires some basic configuration data in order to complete the task successfully.

‘Source and Sink Ref’ identifies the location and type of both source and sink such that a proper implementation for the data transport can be chosen. In our example scenario this is an FTP transfer.

- Format *Conversion*: In this step transported dataset is converted into XML format. In order to perform these types of conversions, this component also requires a schema for the involved dataset (Fig. 6b). Fig. 6c then shows the converted dataset.
- *Data Operation*: This step performs specific operations on the incoming dataset, for instance some arithmetic calculations. In our case we utilize this step for data filtering since the values of attribute ‘VisibilityOneMinuteAverage’ in PWD dataset that are greater than ‘2000’ are normally assumed to be invalid. An implemented function *pFilter(P,R)* is invoked with parameter values *P=”Visibilty”* and *R=”<=2000”*.

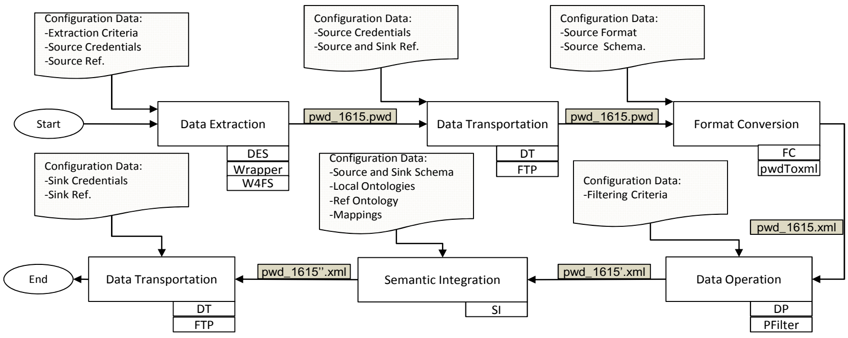
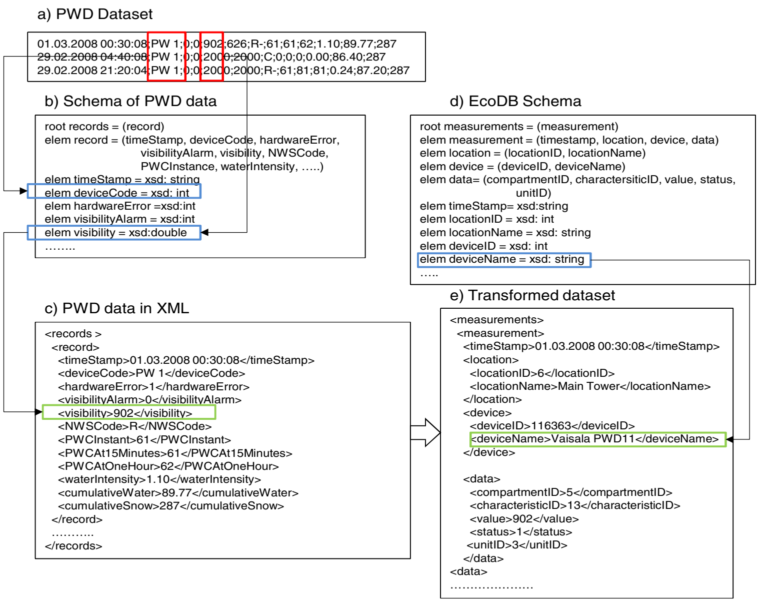


**Fig. 5.** Generated DaLo-WF for data integration in between *Select File* and *Store File*

- *Semantic Integration*: The key purpose of this step is to transform a dataset under one schema into a valid dataset under another schema. In order to detect and resolve semantic conflicts the SI component requires source and sink schemas, respective local ontologies, mappings specifying relationship between local ontologies and data schemas, and a reference ontology. Fig. 6d,e shows the schema of the target database and integrated dataset (i.e. the dataset generated by the SI component) respectively.
- *Data Transportation*: This step is analogous to previous one in which dataset was transported to DaltOn, but at this time datasets are transported from DaltOn to target data source (sink) by invoking relevant component and using specific underlying implemented method. In our example scenario dataset ‘pwd_1615’’.xml’ is shipped to the location where institution database resides.

Finally we want to show how DaltOn solves the issues mentioned in Section 0. The first issue was “Incomplete Information Transportation” which is caused by data conversion and transformation steps since data is transformed via configuration files. Within DaltOn, data are given conformant to a respective ontology. Thus during each action performed by DaltOn, checks can be applied that ensure the completeness of data. Besides SI would throw an error if the data would not be complete at the time the semantic integration is to be performed. The second issue was the missing validation and filtering of data. DaltOn is able to handle data validation and filtering with the help of the DP service. For instance we demonstrated that a specific data item (in the example this was "visibilityOneMinuteAverage") could be filtered out in case its value exceeds a certain range. The condition responsible for the filtering is a configuration item and must not be programmed separately. Another issue discussed in previous section was to manage data formats. Handling of difference formats through a single component (e.g FC) eases the format management overhead in a situation where data are being generated by many heterogeneous sources. The last issue mentioned in the previous section was the addition of new sensors which deliver their data in a slightly different format and ontology. With DaltOn adding new sensors now means to add new semantic descriptions and – if required – a new wrapper for the sensor.



a) PWD Dataset

```
01.03.2008 00:30:08;PW 1;0;0;902;626;R-;61;61;62;1.10;89.77;287
29.02.2008 04:40:08;PW 1;0;0;2000;2000;C;0;0;0;0.00;86.40;287
29.02.2008 21:20:04;PW 1;0;0;2000;2000;R-;61;81;81;0.24;87.20;287
```

b) Schema of PWD data

```
root records = (record)
elem record = (timeStamp, deviceCode, hardwareError,
               visibilityAlarm, visibility, NWSCode,
               PWCInstance, waterIntensity, .....)
elem timeStamp = xsd: string
elem deviceCode = xsd: int
elem hardwareError =xsd:int
elem visibilityAlarm = xsd:int
elem visibility = xsd:double
........
```

c) PWD data in XML

```
<records >
  <record>
    <timeStamp>01.03.2008 00:30:08</timeStamp>
    <deviceCode>PW 1</deviceCode>
    <hardwareError>1</hardwareError>
    <visibilityAlarm>0</visibilityAlarm>
    <visibility>902</visibility>
    <NWSCode>R</NWSCode>
    <PWCInstant>61</PWCInstant>
    <PWCAt15Minutes>61</PWCAt15Minutes>
    <PWCAtOneHour>62</PWCAtOneHour>
    <waterIntensity>1.10</waterIntensity>
    <cumulativeWater>89.77</cumulativeWater>
    <cumulativeSnow>287</cumulativeSnow>
  </record>
  ...........
</records>
```

d) EcoDB Schema

```
root measurements = (measurement)
elem measurement = (timestamp, location, device, data)
elem location = (locationID, locationName)
elem device = (deviceID, deviceName)
elem data= (compartmentID, charactersiticID, value, status,
            unitID)
elem timeStamp= xsd:string
elem locationID = xsd: int
elem locationName = xsd: string
elem deviceID = xsd: int
elem deviceName = xsd: string
.....
```

e) Transformed dataset

```
<measurements>
  <measurement>
    <timeStamp>01.03.2008 00:30:08</timeStamp>
    <location>
      <locationID>6</locationID>
      <locationName>Main Tower</locationName>
    </location>
    <device>
      <deviceID>116363</deviceID>
      <deviceName>Vaisala PWD11</deviceName>
    </device>

    <data>
      <compartmentID>5</compartmentID>
      <characteristicID>13</characteristicID>
      <value>902</value>
      <status>1</status>
     <unitID>3</unitID>
    </data>
<data>
.....................
```

**Fig. 6.** Excerpt of instance data and schemas for both sides along with actual PWD dataset

## 5 Conclusion

In this paper we have shown and discussed a use case for the application of the DaltOn data integration framework. We identified main issues of the approach which is currently applied and that are mainly based on the fact that nearly all portions of the current data integration solution are implemented by hand. With the DaltOn framework for data integration the hand-written code could be replaced and solutions for the mentioned issues can be broken down mostly to the provision of updated semantic information of the data sources and sinks of an integration scenario. Furthermore DaltOn does not force its users to use one specific format or one ontology only. Instead it allows for using those formats and ontologies which are best fitted for the specific application case. Last but not least we have shown that data integration performed with DaltOn is a systematic task. With DaltOn and POPM the scientific workflow design can be separated into two parts; the research oriented task, can be performed by scientists themselves whereas the specification of the data integration, can be performed by a data expert.

## References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York (2003)
2. Rosse, C., Mejino Jr., J.L.V.: A reference ontology for biomedical informatics: the Foundational Model of Anatomy. Journal of Biomedical Informatics 36(6), 478–500 (2003)
3. Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M.R., Li, P., Oinn, T.: Taverna: a tool for building and running workflows of services. Nucleic Acids Research (Web Server Issue) (2006)
4. Jablonski, S., Bussler, C.: Workflow Management – Modeling Concepts, Architecture and Implementation. Intl. Thomson Computer Press, London (1996)
5. Jablonski, S., Volz, B., Rehman, M.A.: A Conceptual Modeling and Execution Framework for Process Based Scientific Applications. In: 1st Workshop on CyberInfrastructure: Information Management in eScience (CIMS), Lisboa, Portugal (2007)
6. Jablonski, S., Rehman, M.A., Volz, B., Curé, O.: Architecture of the DaltOn Data Integration System for Scientific Applications. In: Int'l Workshop on Applications of Workflows in Computational Science (AWCS), Krakow, Poland (2008)
7. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific Workflow Management and the Kepler System. Concurrency and Computation: Practice & Experience 18(10), 1039–1065 (2006)
8. Pei, M., Nakayama, K., Hara, T., Nishio, S.: Constructing a Global Ontology by Concept Mapping Using Wikipedia Thesaurus. In: 22nd International Conference on Advanced Information Networking and Applications - Workshops, 2008. AINAW 2008, pp. 1205–1210 (2008)
9. Present Weather Detector PWD11 User's Guide, Vaisala (1997)
10. World Meteorological Organization, Manual on codes. In: WMO 1995, I.1–C,306, pp.145–147 (1995)
11. Wrzesinsky, T., Scheer, C., Klemm, O.: Fog deposition and its role in biogeochemical cycles of nutrients and pollutants. In: Matzner, E. (ed.) Ecological Studies, Biogeochemistry of forested catchments in a changing environment: a German case study, vol. 172, pp. 191–202. Springer, Heidelberg (2004)

# Experiment Line: Software Reuse in Scientific Workflows*

Eduardo Ogasawara[1], Carlos Paulino[1], Leonardo Murta[2], Cláudia Werner [1], and Marta Mattoso[1]

[1] Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
{ogasawara,kdu,werner,marta}@cos.ufrj.br
[2] Fluminense Federal University, Niterói, Brazil
leomurta@ic.uff.br

**Abstract.** Over the last years, scientists have been using scientific workflows to build computer simulations to support the development of new theories. Due to the increasing use of scientific workflows in production environments, the composition of workflows and their executions can no longer be performed in an ad-hoc manner. Although current scientific workflow management systems support the execution of workflows, they present limitations regarding the composition of workflows when it comes to using different levels of abstractions. This paper introduces the concept of experiment line which is a systematic approach for the composition of scientific workflows that represents an in-silico experiment. An experiment line is inspired on the software engineering reuse discipline and allows the composition of scientific workflows at different levels of abstractions, which characterizes both the in-silico experiment and different workflow variations that are related to the experiment.

**Keywords:** scientific experiment, experiment line, software reuse, scientific workflows, product line.

## 1 Introduction

The evolution of computer science in the last decade enabled the exploration of new types of scientific experiments based on computer simulations, which are commonly known as in-silico experiments. With the performance improvements of computers, it was possible to increase the complexity of the models used in scientific experiments. Researchers perform many activities during these experiments, and some of them are related to the chaining of a sequence of programs. Each program execution may produce a collection of data with a certain semantic and syntax. This data collection can be used as input to the next program to be executed.

The chaining of these programs is not a trivial task, and in many cases, it becomes a barrier to build more sophisticated analyses or models. The term scientific workflow is used to describe workflows [1] in any science area such as biology, physics, chemistry, ecology, geology, and astronomy. These areas share common characteristics

* This work was partially sponsored by CNPq.

like manipulation of large volumes of data and high performance computational demands [2].

Scientific Workflow Management Systems (WfMS) are software packages that provide an infra-structure to setup, execute, and monitor scientific workflows. These WfMS are responsible for coordinating the invocation of programs, either locally or in remote environments, which is commonly known as orchestration. Several WfMS have been developed [2], each one presenting specific characteristics such as strategies for parallel and distributed processing, different primitives for data access (e.g., XML and Database Management Systems), and different scientific and statistical packages.

It is important to note that WfMS are tools to support the scientists in achieving their goals via scientific workflows. Nevertheless, a scientific experiment is characterized by the composition and execution of several variations of workflows. These variations include changing input data, parameters, programs, or even a combination of all of these. WfMS are focused in supporting the execution of a workflow in an isolated way, disregarding the relationship between executions of their variations.

WfMS like Taverna [3], Kepler [4], and VisTrails [5], offer rich graphic interfaces where previously registered components can be dragged and dropped to a workflow editing area. This results directly in the setup of a concrete workflow. However, there is no support to the previous steps of the workflow composition process. Indeed, the composition of a workflow includes the conception of activities, the selection of adequate programs or components to enact these activities, and also the setup of the activities flow. In current WfMS, the action of discovering an activity is limited, since not all components are necessarily registered in the WfMS. Also, the knowledge of which activities can be linked to each other is still tacit. It is necessary to run a large number of examples to gain some experience in the setup of the activity flow.

An alternative, often used by scientists, is to try to reuse a previously defined workflow. However, supporting reuse in WfMS is also limited to a concrete level. Some initiatives have begun to support the setup step of an activity flow of the workflow composition process, such as in VisTrails [5,6] and myExperiment [7]. Based on previously developed workflows, VisTrails suggests a list of related activities to the workflow being setup, but the suggestions are limited to previously executed workflows. The myExperiment [7] initiative provides an interesting site with a repository of previously defined workflows. Most of these workflows are defined under the Taverna workflow definition language and belong to the bioinformatics domain. This workflow repository is very useful when the scientist needs perfect matches. However, adapting this workflow or using a different language for composition is far from trivial.

Currently, projects in bioinformatics and oil and gas domains force scientists to redefine, almost from scratch, scientific workflows previously developed by other scientists, incurring in the same composition trial and error. This occurs due to the absence of a systematic approach and support for the workflow composition.

Meanwhile, in the last four decades, software engineering has been studying systematic ways to support the conception and usage of software. Particularly, software reuse have been applied with success in many organizations, leading to a decrease in rework, and consequently, leading to an increase in productivity and quality [8]. Recently, Roure et al. [9] and Goderis et al. [10] observed the increase of interest in the workflow reuse subject. Although the concern has increased, leading to a wide use of the expression "workflow reuse", it was not possible to observe any related work that analyzes scientific workflow composition with the perspective of software reuse.

This paper introduces the concept of experiment line, which is a strategy to support the composition of scientific workflows at different levels of abstractions. It represents both the in-silico experiment and also different workflow variations that are related to the experiment. An experiment line is inspired by software product lines [11], which is a technique from the software reuse discipline. This work is part of a Brazilian project for supporting large scale management of scientific experiments [12].

This paper is organized into four sections besides this introduction. Section 2 presents an overview of experiment life cycle and the current limitations of WfMS in supporting the composition of experiments. Section 3 discusses the limitations of WfMS in supporting workflows at different levels of abstraction. Section 4 introduces the concept of experiment line. Section 5 concludes the paper.

## 2 Supporting the Life Cycle of Scientific Experiments

A scientific experiment is one of the ways used by the scientific method to support the formulation of new theories. A scientific workflow is the part of a scientific experiment responsible for orchestrating a sequence of processes that manipulate data in order to build a simulation. The life cycle of a scientific experiment starts with a workflow composition, then its execution and further analysis of the results of this execution. The workflow composition process is complex and several trials of workflow variations need to be performed to obtain the desired result of the whole experiment.

Scientific experiments, independent of the application domain, need to follow some requirements, such as:

- Experiments need to be reexecuted and disseminated, allowing other scientists to conduct similar experiments to confirm (or refute) the obtained experiment results. Also, these results need to be documented and can be used as a baseline for other experiments;
- Experiments must follow a protocol or a methodology. They can be started from scratch or from previously obtained results;
- Experiments need to be executed under controlled conditions. Typically, an experiment is an ordered composition of complex steps. These steps need to be documented and controlled during the experiment, allowing other re-executions, if needed;

According to Oinn et al. [13], the life cycle of a scientific experiment has five stages expressed in **Fig. 1**. We clustered these stages into three general workflow phases, which are composition, execution, and analysis. The composition encompasses the conception of an activity, the selection of an adequate program or component to enact the activity, and also the setup of an activity flow for each workflow trial. The execution phase focuses on the execution of workflows, including data and program distribution and monitoring. The analysis phase focuses on the evaluation of

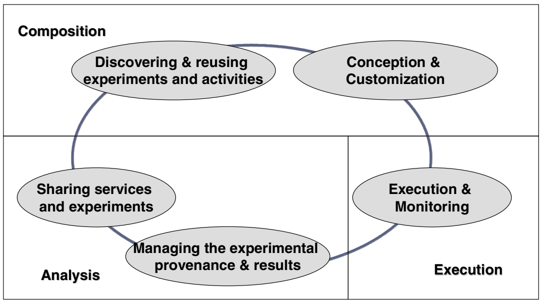

Composition
Discovering & reusing experiments and activities
Conception & Customization
Sharing services and experiments
Execution & Monitoring
Managing the experimental provenance & results
Analysis
Execution

**Fig. 1.** The experiment life cycle adapted from [13] and grouped into three phases in [12]

experimental results obtained from the execution of workflows, which includes activities such as data visualization, queries, and provenance [14].

In the composition phase, which is the scope of this work, an experiment can start from previously designed experiments (*Discovering & reusing experiments and activities*) or from scratch (*Conception and Customization*) using a documented protocol. Whether the experiment is started from scratch or from previously designed experiments, workflow definitions are built or adapted, respectively.

## 3 Limitations of WfMS in Supporting Workflows in Different Levels of Abstractions

There are several abstraction levels in which a workflow can be defined. Frequently, workflows are considered in only two levels [15], either as abstract or concrete. Abstract workflows are specified without defining the resources to be used during execution, leading to flexibility, since it would not be necessary to go into implementation detail. This usually means that workflows are defined as the chaining of conceptual activities. Activities in an abstract workflow are called abstract activities. Concrete workflows specify technological characteristics and define computational resources required to execute the activities. This usually means that the concrete workflows are defined as the chaining of programs. Thus, a concrete workflow is a specific instantiation of an abstract workflow for a particular problem and includes the definition of programs and input data [2]. Activities in a concrete workflow are called concrete activities.

In order to facilitate workflow composition, WfMS need to support different levels of abstraction. However, most current WfMS, such as Taverna [3], only support workflows at concrete level. This kind of support is far from solving the composition problem. To clarify this characteristic, **Fig. 2**.a shows a bioinformatics workflow according to the Taverna notation [16]. Although this paper presents several workflows using the Taverna notation, the discussion presented here transcends a particular WfMS implementation. In Taverna, processes are represented by light gray and dark gray rectangles. Input data are represented by a small upside triangle. Output data are represented by a small downside triangle. To bring a uniform vocabulary among all scientific workflows, Taverna processes will be called activities. These activities are directly linked to each other, and also represent relationship dependence. At concrete level, workflow activities are specific packages or programs of an application domain (light gray activities, for example, *runKalign*) or special adaptor activities (dark gray rectangles, for example, *Unpack_alignment*), used to manipulate an income data and transform it into a correct format in order to feed other activities. In this way, activities have input ports and output ports that are used to interconnect them.

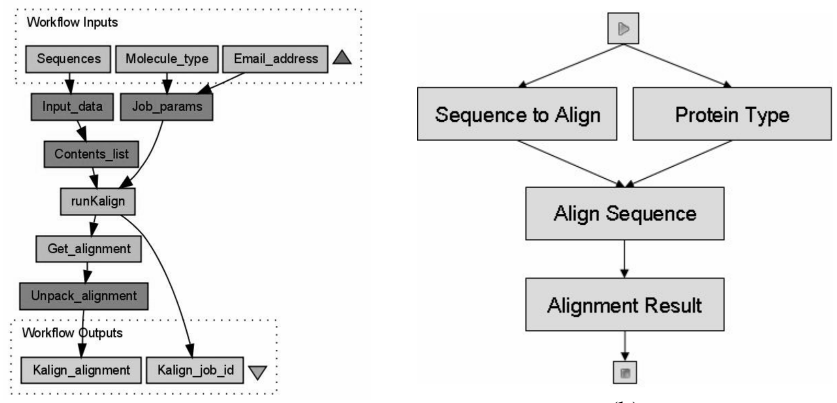


**Fig. 2.** A concrete workflow for sequence analysis with KAlign program (a) and an abstract workflow for the same problem, conceived using the GExpline tool (b).

**Fig. 2**.b presents the same workflow at abstract level. It is simpler to understand a workflow at abstract level when compared to an equivalent concrete level, with all those auxiliary activities. It is clearer at abstract level that it is possible to execute a sequence alignment from a sequence of proteins. An abstract activity establishes that something needs to be done, but does not say how it should be done. Making an analogy with software development [17], an abstract workflow corresponds to software analysis and a concrete workflow corresponds to software design.

Additionally, an abstract activity can be implemented by more than one activity at concrete level. This is also the case of the *alignSequence* activity of **Fig. 2**.b, which at concrete level is implemented by a sequence of three input adaptors (*Input_data*, *Job_params*, and *Contents_list*) necessary to prepare a web service invocation to the *runKAlign* web service, and the two output adaptors (*Get_alignment* and *Unpack_alignment*) of the Taverna workflow of **Fig. 2**.a.

During the scientific experiment, two interesting situations can occur. In the first situation, a scientist may need to run the experiment repeatedly, changing some input data and analyzing the behavior of the model according to the change. In this case, the scientific workflow just need to be executed again with parameter changes [7] using a WfMS. In the second situation, the scientist may not be satisfied with the obtained results and may want to explore different programs to execute his scientific experiment. This may lead to change the workflow to explore different alternatives and different programs to achieve the experimental result. WfMS does not present primitives to help the exploration of variations that express scientific experiment life cycle. This type of change is normally done by making copies of a workflow followed by specific editing. This approach has limitations regarding scalability and maintenance [18], since it loses semantic. For example, if WfMS had support for abstract workflows, the scientist would have known that from the a*lignSequence* abstract activity of **Fig. 2**.b there are many different programs that can execute sequence alignment (*clustalw*, *kalign, mafft, muscle*, and *tcoffee*), and he would just need to select the desired program. Unfortunately, existing WfMS consider any change as a new workflow. Even when there is a clear knowledge of the existence of different programs to run the same activity, there is no formal relationship between them. Moreover, if the scientific experiment needs to expand the analysis scope, it would be necessary to change all scientific workflows related to the experiment, despite the fact that they belong to the same abstract workflow.

## 4 Experiment Line

Experiment line is an approach to represent an in-silico experiment. An experiment line can be characterized by abstract workflows that are capable to be derived into multiple workflows at concrete level. An experiment line is inspired by software product lines [11]. Software product lines are software engineering methods, tools, and techniques for creating a collection of similar software systems from a shared set of software assets using common means of production [11].

Software product lines were created for software development. The success of a product line is related to the way reusable assets are planned. In order to adapt the concepts of product lines to scientific workflows, the first step is to consider an activity as a component. A component is a composition unity that contractually specifies interfaces and explicitly presents its dependencies. In software engineering, there are some works that compare activities in a process with components [19]. This analogy also makes sense when it comes to workflows. Moreover, this component behavior is an important concept to allow the definition of an abstract workflow and its mapping to a list of concrete workflows related to it.

An experiment line is the chaining of activities in an abstract workflow where each activity behaves like a component. Each abstract activity can be implemented by a list of compatible sequences of concrete activities. Also, a sequence of abstract activities can be grouped together to form another abstract activity. When an abstract activity has more than one sequence of abstract activities to establish its behavior, it is called variant activity. Actually, this means that an abstract activity is a variant activity if it has more than one program to implement its conceptual component behavior. Also, when an abstract activity can be suppressed from a derived workflow related to the experiment line, it is defined as an optional activity. A mandatory activity is an abstract activity that must be used in all derived workflows.

Also, the relationship among abstract activities establishes interdependency. Abstract activities are linked together by means of input and output ports, just like concrete activities are linked in a concrete workflow. The way in which the abstract activities are connected, such as using different ports, may affect the behavior of the abstract activity. So, if a relationship between two abstract activities in an experiment line can be connected by more than one way, it is called a variant relationship. Also, if a relationship between two activities can be suppressed from an abstract workflow, it is called an optional relationship at the corresponding experiment line.

The process from which a concrete workflow is obtained from the experiment line is called derivation. A concrete workflow is derived from the experiment line by:

- Choosing one of the abstract activities from each variant activities;
- Choosing if each optional activity is going to be included in the derived workflow;
- Choosing the way in which each variant relationships can link two abstract activities;
- Choosing if each optional relationship is going to be included in the derived workflow.

**Fig. 3** presents an example of an experiment line, with some concrete workflows derived from it. In the example, A, B, and E are mandatory abstract activities, and their sequence of concrete activities are presented in all derived workflows represented by (a, b, e). In this example, D is an optional activity. It is presented in the derived workflow 1, represented by the sequence of concrete activities ($d_1$-$d_2$), but it is not presented in the derived workflow 2. Also, in this example, C is a mandatory variant activity. It has three possible abstract activities to be used during the derivation process ($C_1$, $C_2$, $C_3$), which in turn are implemented by their respective concrete activities ($c_1$, $c_2$, and $c_3$). Since C is also mandatory, all derived workflows must have one of the possible sequences of concrete activities.

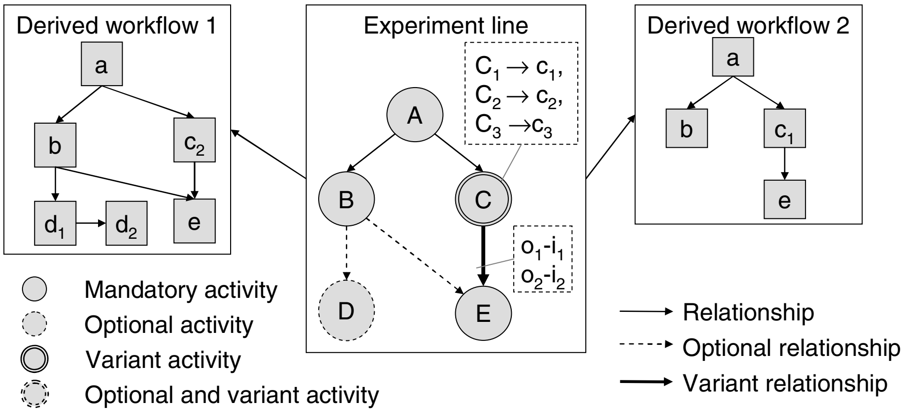


**Fig. 3.** Experiment line and some derived workflows

In addition, in **Fig. 3** there is an optional relationship, from activity B to activity E. This means that the workflow may optionally have a connection between (b) and (e). Also, in the example, the relationship between C and E can be structured in more than one way (i.e., $o_1$-$i_1$ or $o_2$-$i_2$). Depending on the ports chosen to link these two activities, the behavior of the derived workflow can vary.

During or after the derivation process, it is necessary to check if the derived workflow is valid. So, it is necessary to make a formal verification of the derived workflow to guarantee its integrity, which means that the derived workflow needs to start from an initial state and reach a final state. Moreover, the selected activities must be compatible. This process is defined as verification of workflow derivation.

## 5 Conclusions

Scientific workflows represent a first step to the management of scientific experiments. Although much evolution occurred in WfMS, there are still a large number of open problems related to the support of the experiment life cycle. Particularly, the workflow composition in current WfMS is limited to the setup of the activity flow of concrete workflows, which is usually associated with a trial in an experiment. By using concepts from software engineering, it is possible to empower the composition of scientific workflows for WfMS, which allows the characterization of the scientific experiment using both an abstract workflow and all the alternatives concrete workflows that are related to it.

This work introduced the concept of experiment line as a software engineering approach to help the composition and management of scientific experiments. It supports the definition of an experiment line, which includes an abstract workflow, as well as the derivation of concrete workflows that can be obtained from it. In this approach it is possible to allow scientists to compose a workflow using guided information obtained by the scientific experiment represented in an experiment line. It also improves maintenance efforts, since scientists just need to work on the experiment line and let these changes be propagated to the corresponding abstract and concrete workflows during the derivation process.

As a proof of concept, a tool named GExpline is being developed to manage the experiment line. GExpline allows the composition of scientific workflows to be further executed by Kepler [4] and Taverna [16] WfMS. Support to the VisTrails [5] WfMS is under development.

The authors are currently designing an experiment line in the oil and gas domain to build scientific workflows for experiments regarding risers fatigue control in offshore platforms. A semantic representation such as experiment line is mandatory in this scenario, since there is a large number of variable and optional activities that can be chosen by engineers to intensively run their scientific experiments.

## References

[1] Hollingsworth, D.: Workflow Management Coalition: The Workflow Reference Model. The Workflow Management Coalition (1995)

[2] Deelman, E., Gannon, D., Shields, M., Taylor, e. I.: Workflows and e-Science: An overview of workflow system features and capabilities. Future Generation Computer Systems (July 2008)

[3] Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M.R., Li, P., Oinn, T.: Taverna: a tool for building and running workflows of services. Nucleic Acids Research 34 (Web Server issue), 729–732 (2006)

[4] Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., Mock, S.: Kepler: an extensible system for design and execution of scientific workflows. In: Proceedings. 16th International Conference on Scientific and Statistical Database Management, Santorini, Greece, pp. 423–424 (2004)
[5] Koop, D., Scheidegger, C., Callahan, S., Freire, J., Silva, C.: VisComplete: Automating Suggestions for Visualization Pipelines. IEEE Transactions on Visualization and Computer Graphics 14(6), 1691–1698 (2008)
[6] Oliveira, F., Murta, L., Werner, C., Mattoso, M.: Using Provenance to Improve Workflow Design. In: Second International Provenance and Annotation Workshop - IPAW, Salt Lake City, UT, USA, pp. 136–143 (2008)
[7] Goderis, A., De Roure, D., Goble, C., Bhagat, J., Cruickshank, D., Fisher, P., Michaelides, D., Tanoh, F.: Discovering Scientific Workflows: The myExperiment Benchmarks. IEEE Transactions on Automation Science and Engineering (2008)
[8] Frakes, W., Kang, K.: Software reuse research: status and future. IEEE Transactions on Software Engineering 31(7), 529–536 (2005)
[9] Roure, D.D., Goble, C., Stevens, R.: Designing the myExperiment Virtual Research Environment for the Social Sharing of Workflows. In: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, Bangalore, India, pp. 603–610 (2007)
[10] Goderis, A., Sattler, U., Lord, P., Goble, C.: Seven Bottlenecks to Workflow Reuse and Repurposing. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 323–337. Springer, Heidelberg (2005)
[11] Northrop, L.: SEI's software product line tenets. IEEE Software 19(4), 32–40 (2002)
[12] GExp, Large Scale Managament of Scientific Experiments (2009), `http://gexp.nacad.ufrj.br/`
[13] Oinn, T., Li, P., Kell, D.B., Goble, C., Goderis, A., Greenwood, M., Hull, D., Stevens, R., Turi, D., et al.: Taverna/myGrid: Aligning a Workflow System with the Life Sciences Community. In: Workflows for e-Science, pp. 300–319. Springer, Heidelberg (2007)
[14] Freire, J., Koop, D., Santos, E., Silva, C.T.: Provenance for Computational Tasks: A Survey. Computing in Science and Eng. 10(3), 11–21 (2008)
[15] Yu, J., Buyya, R.: A Taxonomy of Workflow Management Systems for Grid Computing. Journal of Grid Computing 34(3-4), 171–200 (2005)
[16] Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., et al.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Oxford Univ. Press, Oxford (2004)
[17] Pressman, R.S.: Software Engineering Software Engineering: A Practitioner's Approach, 6th edn. McGraw-Hill, New York (2004)
[18] Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M.: Workflows for e-Science: Scientific Workflows for Grids, 1st edn. Springer, Heidelberg (2006)
[19] Fusaro, P., Visaggio, G., Tortorella, e.M.: REP - ChaRacterizing and Exploiting Process Components: Results of Experimentation. In: Proceedings of the Working Conference on Reverse Engineering (WCRE 1998), pp. 20–29 (1998)

# Tracking Files in the Kepler Provenance Framework

Pierre Mouallem[1], Roselyne Barreto[2], Scott Klasky[2], Norbert Podhorszki[2], and Mladen Vouk[1]

[1] North Carolina State University, 890 Oval Dr, Raleigh, NC 27695
{pmouall,vouk}@ncsu.edu
[2] Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, TN 37831
{barreto,klasky,pnorbert}@ornl.gov

**Abstract.** Workflow Management Systems (WFMS), such as Kepler, are proving to be an important tool in scientific problem solving. They can automate and manage complex processes and huge amounts of data produced by petascale simulations. Typically, the produced data need to be properly visualized and analyzed by scientists in order to achieve the desired scientific goals. Both run-time and post analysis may benefit from, even require, additional meta-data – provenance information. One of the challenges in this context is the tracking of the data files that can be produced in very large numbers during stages of the workflow, such as visualizations. The Kepler provenance framework collects all or part of the raw information flowing through the workflow graph. This information then needs to be further parsed to extract meta-data of interest. This can be done through add-on tools and algorithms. We show how to automate tracking specific information such as data files locations.

**Keywords:** Data Tracking, Data Provenance, Scientific Data Management, Scientific Workflows.

## 1 Introduction

Managing complexity and volume of data has been identified as one of the most important emerging needs by the scientific community. A very significant component is efficient generation and handling of the meta-data.

The Scientific Process Automation group (SPA) [1] of the DOE Scientific Data Management Center (SDM) [1] is researching, developing and deploying data management tools. Kepler workflow management system [2] is one such open-source tool based on the PTOLEMY II [3] framework. In a Kepler workflow, a process is called an actor. Actors are interconnected by communication channels through which the data flow in the form of tokens. Execution of the whole workflow is controlled by one of a number of special schedulers called Directors. SDM has also developed a provenance collection framework for Kepler [4], along with visual interfaces to display and analyze the results – the dashboard [5]. The Kepler provenance recording mechanism “listens” to the token flows to collect meta-data. A challenge is what and how much to collect. One the one hand provenance data needs to be sufficiently fine grained to be useful, and on the other hand collecting too much information is an overhead that we want to avoid.

While tracking input and output files is a relatively routine activity during computations, it may become a challenge when the volume of data being generated is large, and there is a need to analyze that data quickly, When there is a large volume of meta-data this information may be buried and hard to get to. Thus efficient mechanisms that that take advantage of the provenance data to provide file tracking capabilities, with minimum overhead to the end user, are of interest.

In section 2 of this paper we describe the provenance framework and the provenance viewing mechanism we use with Kepler. In section 3 we discuss ad-on algorithms for tracking data files. In section 4 we briefly discuss related work, and section 5 concludes the paper.

## 2 Provenance

Data provenance refers to the data about the origin and the history of the data, and its transformations and derivatives. This information can be used to track evolution of the data, and gain insights into the analyses performed on the data. Provenance of the processes, on the other hand, enables scientist to obtain precise information about how, where and when different processes and operations were applied to the data, where and when failures or warning occurred, where the data was stored, etc.

### 2.1 Provenance Framework

The Kepler provenance framework developed by the SPA team [6] is illustrated in Figure 1. At the core is the provenance database with its APIs. The generality of the design allows insertion of another database (or another storage option) of choice. API has three key components: (1) Kepler, its actors, and external scripts use a **Recording API** to collect and save provenance information; (2) a **Query API** provides different query capabilities for accessing the data; and (3) a M**anagement API** for Provenance Store maintenance. Currently, the recording API is actually part of Kepler. We are in the process of writing a separate, detachable, API affiliated with the storage solution.

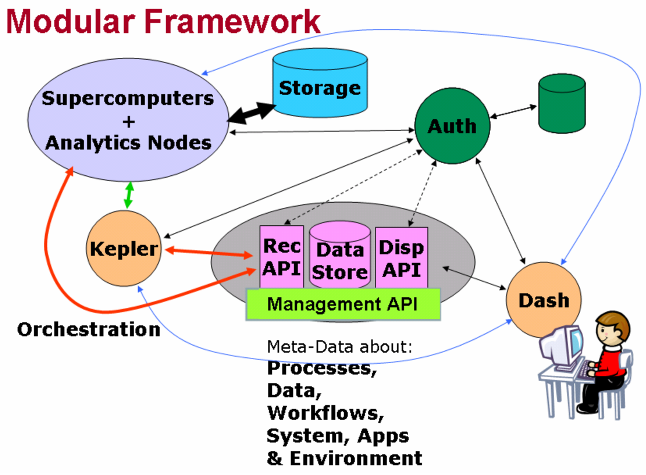


**Fig. 1.** Provenance Framework

The database stores all provenance information. The database must be self-contained, but it can be distributed. Simple items like annotations, user identifiers, timestamps, etc., are stored in closely corresponding SQL data types, such as numeric or varchar. More complex items are stored as BLOBs. For example, the most general type of a Kepler token (item passed between actors) can be a Java Object. In this case, if the user wishes to save all tokens produced by the actors in his workflow, he/she would need to provide serialization routines to store and retrieve the relevant Java Objects. An alternative is to use database stored pointers to reference the provenance information outside the database, such as Java Objects or workflow configuration files, pictures, movies or application source code. However, this solution may require a mechanism to reference data across machine boundaries. Self-containment guarantees that modifications to provenance information are done only using the Provenance Store Recording API.

The Kepler workflow management system implements a Provenance Recorder [4, 7]. It is a set of listeners, or hooks – that saves token-based provenance from all (internal) workflow components into the Provenance Store. Depending on the granularity, that data may be recorded for all actors in the workflow, or for some subset, e.g., only top-level composites. The **recording API** additionally supports components external to Kepler. These components are usually Python or shell scripts called by actors running in a workflow. Furthermore, these scripts may execute on a machine other than the one running Kepler. An example is the performance parameters that may come from software instrumentation and performance tools. In this paper we are particularly interested in the information stored about actor firings and tokens, which will be the basis of the analysis routine mentioned later.

The **Query API** provides a read-only mechanism to retrieve provenance information from the database. It provides a range of capabilities from notifying applications as provenance is recorded (e.g., a web based dashboard) to querying details about past executions. In addition to providing current workflow status, applications can query the Provenance Store about past executions. The Query API contains an SQL interface to support these types of queries; it can retrieve data from the database given appropriate authorization. Finally it also provides a callback mechanism for applications wishing to receive real-time provenance updates.

The **management** interface provides user administration and maintenance operations. User administration includes, adding and deleting users, modifying user passwords, modifying user access rights, and specifying the set of accessible workflows.

**Security** of the system has always been of importance. For example, in the current context of its use, the challenge is automated communication and exchange of data with other government labs. We are in the process of implementing a certificate-based security envelope that will allow inter-laboratory exchanges. Furthermore, we are increasingly concerned about the sharing of the provenance data. We realize that workflow meta-data and provenance information may have as much value as the raw information. Typically, sensitive information produced by a computational processes or experiments is well guarded. However, this may not necessarily be true when it comes to provenance information. The issue is how to appropriately share confidential provenance information. We developed a model for sharing provenance information when the confidentiality level is dynamically decided by the user [8].

Also needed is an intuitive user-friendly interface so that the scientists can monitor, query, and access provenance data and analyses [5, 9], Dashboards, as their name implies, are data and information display devices or interfaces that typically present condensed information about the status of workflow processes, data, environment, and so on. The current version of the dashboard has 2 major functionalities: Machine Monitoring and Simulation Monitoring. The former tracks jobs, resources being used, and so on. The latter provides an analysis tool to view and analyze the data being computed. In this case, the views consist of two dimensional plots. Each of the pictures is associated with a file, and path to that information is collected by the provenance component of the system.

## 3 Tracking Data Files

The amount of data collected by Kepler provenance recorder can be very large, and of course it grows with the size and complexity of the workflow. Usually the information of interest is only a small fraction of what is being collected. Having the scientists search through that data is time consuming, and the task of mining the provenance data needs to be automated. For example, if a scientist wishes to further investigate or analyze an image that's being displayed on the dashboard, he/she may need to first locate the file that contains the data behind that image. This is not an evident task since a single run can have hundreds of images associated with it. Furthermore, the relationship between images and the data files may not be one-to-one. This means that one data file may contain the data for several images. Consequently the scientist needs to both locate the data files on the disk, and to examine the contents of the data files before it can be determined which one contains the data of interest. That can generate a lot of overhead, especially for complex workflows which have multiple codes integrated in it, such as those discussed in [10, 11].

To illustrate development of provenance meta-data analysis algorithms, and their insertion in to our Kepler-based provenance framework, we use image data file tracking and archiving as an example.

### 3.1 Tracking Files That Exist on Disk

A solution that was developed early involved tracking of the tokens consumed by the actors that are responsible for generation of the images. This works, however it is not easily ported from one workflow to another since the names of the actors differ among workflows. For example, for one workflow we could be monitoring actor X because we know that it produces the images that we're tracking. Applying that solution to a different workflow might not work because the actor generating those images might have a different name, or other identifying characteristics, etc... It is not unusual to have more than one actor generating images. So an actor independent solution is needed.

That more generic solution does not rely on the actor names, instead, it only checks the tokens generated and consumed, and creates a reverse graph of the token flow, starting with the image name and ending with the data file(s). To better understand this solution, let us consider the simple example in figure 2. It shows a portion of a

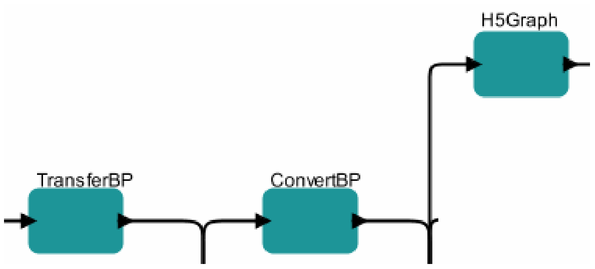


**Fig. 2.** Actors that generate image file

Plasma Edge Simulation workflow [10]. The "ConvertBP" actor takes as input a BP[1] file and generates an HDF5 file, and the "H5Graph" uses the HDF5 file to generate an image (in png format).

Using algorithm 1 (written in PHP), a graph backtracking algorithm similar to the ones found in [12],, we can retrieve the data files that were used to generate the given image. The algorithm would be used by the dashboard code, or could be part of a script that queries the database through the display API.

```
program RetreiveDataFiles(ImageFileName):
  tokenId = getTokenContaining(ImageFileName);
  fireId = getFireId(tokenId);
  inputTokens[] = getConsumedtokens(fireId);
  foreach token in inputTokens[] do
    if token.contains(requestedDataFile) then
      return Token;
    elseif token.contains(intermediateDataFile) then
      RetreiveDataFiles(intermediateDataFile):
    end
end
```

**Fig. 3.** Algorithm 1

Once the data files paths are retrieved, we should be able to draw a graph as in figure 4 that represents the data transformation and image generation. The graph can be described as a reverse token flow.

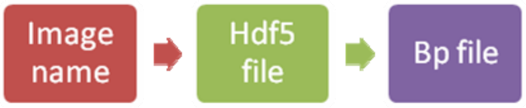


**Fig. 4.** Token Trace of image Files

This method has proven to be effective with a multitude of workflows that follow the same structure as the one mentioned above. It has been extended to provide several alternatives to the query mentioned in the example above. Those alternatives

[1] BP is a special data format developed for the workflow to speed up I/O operations [10].

include retrieving the data files that generated a movie (a collection of images that represent different timesteps), retrieving all data files for all images at a particular timestep, or retrieving all the data files for all timesteps.

One issue that we did notice is performance. Based on the size of the workflow, the database query would sometimes take as much as a minute to execute (in our case we are using MySQL database). Now, since that information for a particular workflow run would not change (once the run has finished executing), and since the scientist will probably be querying for the same data more than just once, we decided that it would be efficient to store the query results in a separate table that simply links images to data files, variables and timesteps. Algorithm 2 illustrates the new approach.

```
program RetreiveDataFiles2(ImageFileName):
  dataFile = queryResultsTable(ImageFileName);
  if dataFile.found() then
    return dataFile;
  else
      return RetreiveDataFiles(ImageFileName);
  end
end
```

**Fig. 5.** Algorithm 2

Algorithm 2 is much simpler and requires a fraction of the time needed by Algorithm 1 to execute. The SQL query in Algorithm 1 contains several joins and several regular expressions evaluations. The query in Algorithm 2 is a simple select statement. This solution is cast as a dashboard analysis tool that takes advantage of the data collected by the Kepler provenance framework. It is currently used in production.

## 3.2 Tracking Archived Data Files

In this subsection we discuss an analysis, based on the same token tracing and analysis principles, that solves more a complicated problem - archiving.

Typically, data files generated during a simulation are stored for only a certain period of time on active disk space allocated for that experiment. Then they are deleted. Therefore scientists need an automated way to track archived files and to determine which of the actual archive files contains the data file of interest. Not surprisingly, one tool that the scientists requested in our context was the ability to track backup files that contain the data files mentioned earlier. The workflow archives all the data files generated during a run into TAR files, and then it backups up those TAR files onto a long-term tape-based storage system (HPSS) [13].

Achieving this needed a more complex algorithm. The main reason is that instead of tracing backwards, we need to trace forward through the branching trees, and. the token trace is not linear when tracing forward through the branching trees.

Now, since we want our analysis module to be actor independent, the first task is to trace the tokens over all possible paths and identify the one(s) of interest. This differs from backtracking because it occurs in our case along a linear path. We use a Breadth First Search [12] to trace the tokens along all possible paths. Our algorithm checks the output of the actors that have consumed the specific token of interest to see if it had generated a

**Table 1.** Token Consumption/Generation example for "Archiver" Actor

| FireId | Token Consumed | Token Generated |
|---|---|---|
| 1 | file1.h5 | --- |
| 2 | *file2.h5* | --- |
| 3 | file3.h5 | --- |
| 4 | file4.h5 | file1-4.tar |
| 5 | file5.h5 | --- |

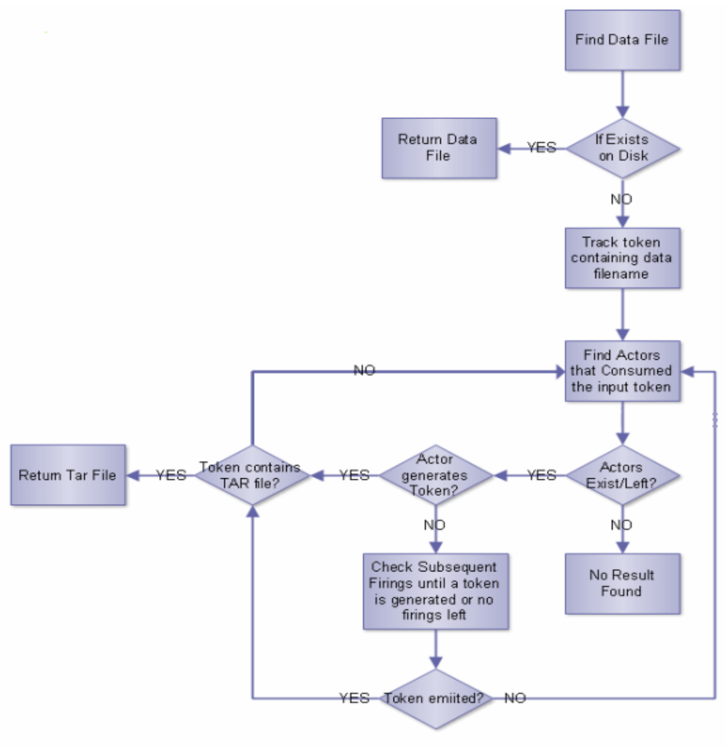


**Fig. 6.** Tracking TAR files

TAR file, or some other type of file. If this actor is found, then we focus on that actor, otherwise we continue recursively until the appropriate actor is found. Once we determine which actor is generating the TAR files, we repeat the same process described in algorithm 1 in order to determine which TAR file contains the data file that we need.

One problem that we faced while implementing this module is that the "Archiver" actor consumes several tokens before producing one. In other words, the data file in question can be consumed without having a TAR file generated for that same FireId (i.e., the id that indentifies the start of the process). Instead, the TAR file would be generated once a sufficient number of data files are consumed. Table 1 illustrates the problem.

For example, if we are looking for the TAR file that contains "file2.h5" and we lookup the FireId, we will not be able to find a result since no tokens were generated for that particular FireId. To solve that problem, during our search for actors that generates TAR files, if we detect one that did not generate any tokens, we search incrementally for all subsequent FireIds that involve that actor until we detect a token being generated. If that token did not turn out to contain a TAR file or no tokens were detected, that means the actor in question is not the one that we're looking for. If it did turn out to contain a TAR file, that would be the TAR file that we are looking for, and it is returned to the end user. This is summarized in figure 6.

## 4 Related Work

A number of provenance models have been proposed in the literature [14, 15, 16, 17, 18, 19]. These models differ in many ways, but the main difference among them is often in the way they use structures and storage strategies. They all share an essential type of information: process and data dependencies.

For example, reference [14] describes a provenance system for data driven workflow called "Karma2." This system uses web services to store and retrieve provenance data from a relational database. However, it does not offer analysis tools for the collected data. Instead, it offers an API through which to collect that data. In references [15] authors use the same technique to collect and retrieve data, except that it differentiates between what they calls "process documentation", which is similar a record of what occurred, and "item provenance" which describes the provenance of a data item. Reference [17] describes a provenance scheme that tracks the provenance during workflow creation and execution. It uses OWL [20] to represent that provenance information. Reference [18] describes a workflow management system (Vistrails) with built-in provenance tracking. It stores the workflow evolution provenance information as an XML tree. In [19] authors address the performance issues in storing provenance data, and propose an alternative approach for storing that data. This work relies on transforming workflows into directed acyclic graphs and storing the data lineage using a tree structure.

On the practical side, a number of teams participating in the provenance challenge [21] provided a series of functions to analyze provenance data. One team, RWS [4, 22, 23] described methods to track the lineage. One of their methods "tokenLineageofValue" provides a list of all tokens that contain a particular value. This algorithm does not establish the token trace that is required to retrieve the data files discussed here, but it is the first step in our analytics.

## 5 Conclusion and Future Work

Provenance based analysis is proving to be a valuable tool and is receiving good feedback from the scientists. In this paper we described some of the Kepler provenance based analysis algorithms currently used in production at ORNL. Specifically tools for tracking and finding files. One algorithm tracks the data files describing images and movies that are being displayed on the dashboard, and the other algorithms tracks the TAR archives that contain those data files in the case those file are no longer available on disk.

However, while solution presented works very well in the case of the workflows we have examines so far, it probably is not general enough. That will be addressed in the future. One of the limitations is that the current solution assumes that only one actor is responsible for generating TAR files. This situation is valid for all of our workflows right now, but might not be true in the future, thus it needs to be addressed.

Although we only discussed tracking of data files, this work opens the door for dealing with much more complex issues. For example, error detection and fault tolerance through backtracking. The provenance data collected about current and old runs has potential to detect errors automatically. We are currently exploring that idea.

## Acknowledgments

We would like to thank our colleagues in the SDM center for useful input, discussion and interactions, Ilkay Altintas, Daniel Crawl from SDSC, and Meiyappan Nagappan from NCSU. This research funded in part by the DOE grant DE-FC02-ER25809 and by the IBM Shared University Research program.

## References

1. Scientific Process Automation (SPA) (2009), `http://sdm.lbl.gov/sdmcenter/`
2. Kepler Project (2009), `http://kepler-project.org/`
3. Eker, J., Janneck, J., Lee, E., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity – the ptolemy approach. Proceedings of the IEEE 91(1) (January 2003)
4. Altintas, I., Barney, O., Jaeger-Frank, E.: Provenance Collection Support in the Kepler Scientific Workflow System. In: International Provenance and Annotation Workshop (IPAW 2006), Chicago, Illinois, USA, May 3-5 (2006)
5. Klasky, S., Barreto, R., Kahn, A., Parashar, M., Podhorszki, N., Parker, S., Silver, D., Vouk, M.: Collaborative visualization spaces for petascale simulations. In: International Symposium on Collaborative Technologies and Systems, pp. 203–211 (May 2008)
6. Altintas, I., et al.: Provenance in Kepler-based Scientific Workflow Systems. In: Poster # 41, at Microsoft eScience Workshop Friday Center, October 13 - 15, p. 82. University of North Carolina, Chapell Hill, NC (2007)
7. Kepler Provenance Recorder Framework (2009), `http://kepler-project.org/Wiki.jsp?page=KeplerProvenanceFramework`
8. Nagappan, M., Vouk, M.: A Privacy Policy Model for Sharing of Provenance Information in a Query Based System. In: Short Paper and Poster in the Second International Provenance and Annotation Workshop (IPAW 2008), Salt Lake City, UT, June 17-18 (2008)
9. Klasky, S., Beck, M., Bhat, V., Feibush, E., Ludäscher, B., Parashar, M., Shoshani, A., Silver, D., Vouk, M.: Data management on the fusion computational pipeline. In: SciDAC 2006, Journal of Physics: Conference Series, vol. 16, pp. 510–520 (2005)
10. Cummings, J., Pankin, A., Podhosrzki, N., Park, G., Ku, S., Barreto, R., Klasky, S., Chang, C.S., Strauss, H., Sugiyama, L., Snyder, P., Pearlstein, D., Ludäscher, B., Bateman, G., Kritz, A.: Plasma edge kinetic-MHD modeling in tokamaks using Kepler workflow for code coupling, data management and visualization. Communications in Computational Physics 4, 675–702 (2008)

11. Chen, J., Choudhary, A., Supinski, B., DeVries, M., Hawkes, E., Klasky, S., Liao, W., Ma, K., Mellor-Crummey, J., Podhorszki, N., Sankaran, R., Shende, S., Yoo, C.: Terascale direct numerical simulations of turbulent combustion using S3D. Computational Science and Discovery 2015001, 31 (2009)
12. Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
13. High Performance Storage System (2009), `http://www.hpss-collaboration.org/hpss/index.jsp`
14. Simmhan, Y., Plale, B., Gannon, D.: Karma2: Provenance management for data driven workflows. International Journal of Web Services Research 5, 1 (2008)
15. Miles, S., Groth, P., Munroe, S., Jiang, S., Assandri, T., Moreau, L.: Extracting Causal Graphs from an Open Provenance Data Model. Concurrency and Computation: Practice and Experience 20(5), 577–586 (2008)
16. Cohen, S., Boulakia, S., Davidson, S.: Towards a model of provenance and user views in scientific workflows. In: Leser, U., Naumann, F., Eckman, B. (eds.) DILS 2006. LNCS (LNBI), vol. 4075, pp. 264–279. Springer, Heidelberg (2006)
17. Kim, J., Deelman, E., Gil, Y., Mehta, G., Ratnakar, V.: Provenance trails in the wings/pegasus system. Concurrency and Computation: Practice and Experience 20(5), 587–597 (2008)
18. Freire, J., Silva, C., Callahan, S., Santos, E., Scheidegger, C., Vo, H.: Managing rapidly-evolving scientific workflows. In: Moreau, L., Foster, I. (eds.) IPAW 2006. LNCS, vol. 4145, pp. 10–18. Springer, Heidelberg (2006)
19. Heinis, T., Alonso, G.: Efficient lineage tracking for scientific workflows. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 1007–1018 (2008)
20. OWL (2009), `http://www.w3.org/TR/owl-guide`
21. Provenance Challenge, `http://twiki.ipaw.info/bin/view/Challenge/`
22. Podhorszki, N., Altintas, I., Bowers, S., Guan, Z., Ludaescher, B., McPhillips, T.: RWS, in First Provenance Challenge (2006), `http://twiki.ipaw.info/bin/view/Challenge/RWS`
23. Bowers, S., McPhillips, T., Ludäscher, B., Cohen, S., Davidson, S.B.: A Model for User-Oriented Data Provenance in Pipelined Scientific Workflows. In: Moreau, L., Foster, I. (eds.) IPAW 2006. LNCS, vol. 4145, pp. 133–147. Springer, Heidelberg (2006)

# BioBrowsing: Making the Most of the Data Available in Entrez

Sarah Cohen-Boulakia and Kevin Masini

Laboratoire de Recherche en Informatique (LRI)
Université Paris-Sud XI, 91405 Cedex Orsay, France

**Abstract.** One of the most popular ways to access public biological data is using portals, like Entrez (NCBI) which allows users to navigate through the data of 34 major biological sources following cross-references. In this process, data entries are inspected one after the other and cross-references to additional data available in other sources may be followed. This navigational process may be time-consuming and may not be easily reproduced from one entry to another. Most importantly, only a few sources are initially queried, biologists do not exploit all the richness of the data provided by Entrez, and in particular they may not explore alternative source paths that provide complementary information.

In this paper, we introduce BioBrowsing, a tool providing scientists with access to the data obtained when all the combinations between NCBI sources have been followed. Querying is done on-the-fly (no warehousing). As new sources and links between sources appear in Entrez, BioBrowsing has a module able to update automatically the schema used by its query engine. Finally, BioBrowsing makes it possible for users to define profiles as a way of focusing the results on users specific interests.

**Availability:** http://bioguide-project.net/biobrowsing

## 1 Motivation

Faced with the deluge of raw data produced by high-throughput technologies and stored in some major public genomic sources (or data banks), scientists (either individually or as consortia) frequently organize and (re-)interpret these data to create *curated* data sources. These sources contain *entries*, that is, annotated files drawn from scientific publications and commented on by curators. As a consequence, these entries reflect the opinion of their curators and contain highly complementary and possibly divergent data, since scientific experts may disagree. Moreover, the annotated files frequently reference entries in external data sources using hypertext links, called *cross-references*, making it possible to find additional information from one entry in a given source to another. Biological data thus form an intricate network of valuable data linked by cross-references. As a result, the number and size of sources providing specialized biological information have increased exponentially over the past ten years; more than one thousand public data sources are currently counted [1].

One of the most popular ways to access public biological data [1] is using portals like Entrez[1], Expasy[2], or SRS [2] which give users the ability to pose keyword-based queries and navigate using cross-references through the data obtained. Entrez, developed by the NCBI (National Center for Biotechnology Information), is probably the most widely used portal, giving access to 34 major biological public sources including RefSeq, PubMed, and OMIM to cite only a few of them.

More importantly, NCBI sources and links between NCBI sources are changing over time. First, new sources may be added to the set of sources available through Entrez, such as the OMIA database added in January 2006. Second, several (independent) sources may be created as a subpart of a major source, such as the Books and Journal sources which are fragments of PubMed. Third, sources may be modified or their names changed such as LocusLink renamed as EntrezGene in March 2005. As a consequence of all these changes, links between NCBI sources are created and deleted.

To illustrate the navigational process followed when Entrez is queried, let us consider the following example based on real use cases we have identified through collaborations with biologists and physicians[3]. Tom, a physician, is interested in knowing the genes involved in the Narcolepsy disease. Tom is used to querying OMIM, the *Online Mendelian Inheritance in Man*, which contains information on genetic phenotypes through entries very carefully annotated by experts. Tom uses Entrez and types `Narcolepsy` as the keyword to be searched. As a result, 19 OMIM entries are found by Entrez. Tom inspects these entries and may follow cross-references from them to the database named *Gene* in Entrez. If this process is done for the 19 OMIM results, 22 distinct genes are found in Entrez Gene. It worth noticing that this step involves not only clicking 19 times on links but also removing duplicates among the set of genes found. At this point, it could be considered that all information about the genes possibly involved in Narcolepsy have been obtained.

However, Tom has missed crucial information. Two other paths between sources would have given him additional information. Indeed, if Tom had considered following the path $OMIM \rightarrow (Entrez)Protein \rightarrow (Entrez)Gene$ (that is, following cross-references from OMIM entries to Entrez Protein entries and then from Entrez Protein entries to Entrez Gene entries) or $OMIM \rightarrow PubMed \rightarrow (Entrez)Gene$, he would have known that a *Nature Genetic* paper (published in late 2008) has shown that the portion between genes CPT1B and CHKB was susceptible to be associated with Narcolepsy. This information existed in Entrez but the user would have had to consider intricate paths between sources to get it.

There is thus a crucial need to help scientists exploit all the information available, and in particular explore alternative source paths that may provide complementary information.

In this paper, we introduce BioBrowsing, a tool providing scientists with access to the data obtained when all the combinations between sources from NCBI have

[1] www.ncbi.nlm.nih.gov/Entrez
[2] www.expasy.ch
[3] More information is available at http://bioguide-project.net

been followed. BioBrowsing poses on-the-fly queries to Entrez (no warehousing) and may be used freely online. As new sources and links between sources appear in Entrez, BioBrowsing has an update module, able to maintain automatically the schema used by its query engine. Finally, BioBrowsing makes it possible for users to define profiles as a way of focusing the results on users specific interests.

## 2 BioBrowsing Architecture

### 2.1 Overview

The BioBrowsing architecture is represented in Figure 1. BioBrowsing relies on two graphs: (i) the *graph of sources* representing the network of sources, composed of sources (e.g., OMIM) and cross-references between them, and (ii) a logical layer, the *graph of entities*, representing the biological entities (e.g., Gene, Disease) and their relationships. These two graphs form the query support.

Queries are expressed by selecting entities of interest and possibly specifying the sources the user wants to be queried.

In the rest of this section, we describe how BioBrowsing generates the query support and then how it is used to answer a user query.

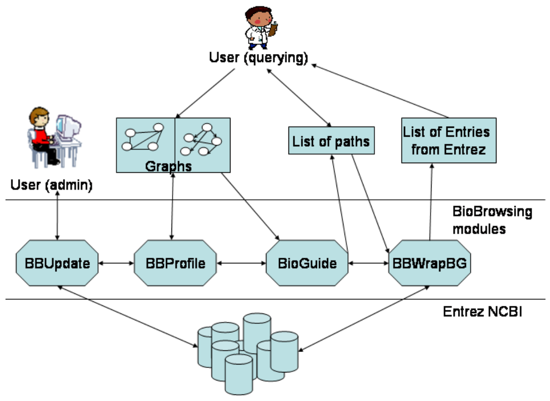


**Fig. 1.** BioBrowsing Architecture

As for the implementation, graphs of sources and entities are expressed within XML files. The BioGuide module has been previously implemented in JAVA [3] while BBUpdate, BBProfile and BBWrapBG modules have been recently developed in Perl. These 3 modules make use of specific functions called *eUtilities*, provided by the NCBI to access their data. Embedded Perl into JAVA has thus also been used to allow the modules to communicate together.

### 2.2 BBUpdate and BBProfile: Generating the Query Support

*Generating the graph of sources.* The BBUpdate module of BioBrowsing takes as input information extracted from Entrez NCBI to create automatically the graph of sources. In the graph of sources each NCBI source is a node ($S_i$) and there is a link from node $S_1$ to node $S_2$ iff there exists cross-references from entries of $S_1$ to entries of $S_2$.

*Generating the graph of entities.* Given the description of the sources provided by the NCBI (a paragraph in natural language), the set of biological entities (Gene, Protein, etc.) available in each source is determined. This step currently involves the participation of the BioBrowsing administrator but could be semi-automatically performed using text mining techniques (e.g., [4]). For example, OMIM contains information on the Disease entity while (Entrez)Gene on Gene. This information is called the *contains* information in our approach. Note that the set of entities to be considered is chosen by the BioBrowsing administrator but can be changed.

From the graph of sources and the *contains* information, BBUpdate generates automatically the graph of entities and the mapping between the two graphs: Nodes of the graph of entities are biological entities ($E_i$) and there is an edge between $E_1$ and $E_2$ iff (i) $E_1$ is contained in source $S_1$, (ii) $E_2$ is contained in source $S_2$, and (iii) there is a link between $S_1$ and $S_2$ in the graph of sources.

*Considering Profiles.* Based on feedback we got from our biologists collaborators, it clearly appears that each user may not be interested in the diversity of data available in NCBI sources. BioBrowsing thus provides *profiles*; given a set of biological entities of interest to be chosen by the user among all the available entities, a sub-graph of the graph of entities is generated as well as a sub-graph of the graph of sources focused on the sources able to provide information on the chosen entities.

Three profiles have currently been predefined: *Medical*, *Annotation* and *Complete*. In our example, Tom may use the *Medical* profile. It is worth noticing that designing a new profile is a matter of minutes and is only based on basic information provided by the user about biological entities of interest to him, there is no need to know anything about the sources.

Average values for graph features are captured in Table 1, depending on the profiles. $TotNBE$ indicates the total number of entities in the graph of entities; $TotNBS$ indicates the total number of sources in the graph of sources; $TotLE$ (respectively, $TotLS$) gives the total number of links in the graph of entities (respectively, sources); $MinL(Ns/e) - MaxL(Ns/e)$ indicates the minimum and maximum number of links between sources providing a given entity while $Mean(Ns/e)$ gives the average value of this number.

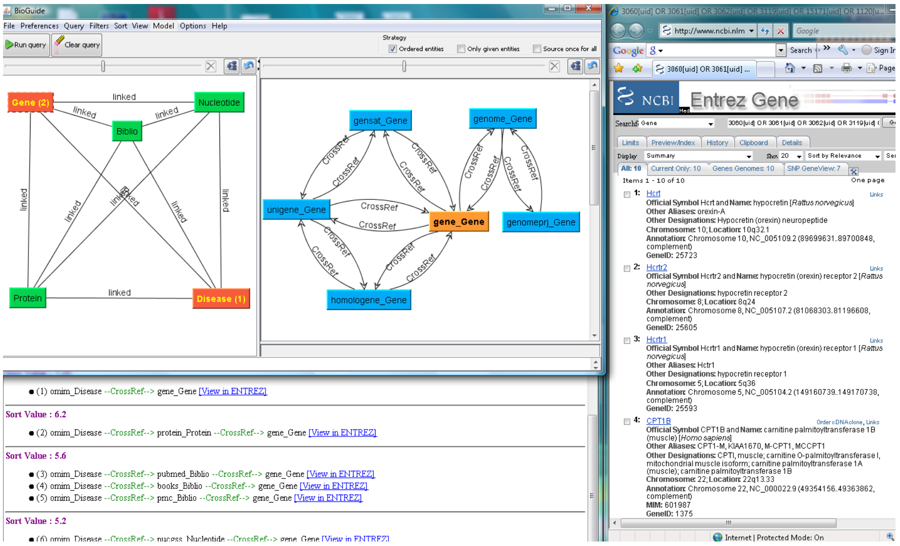
Note that even when a few entities are considered (e.g., in the medial profile), the graph of sources may be huge (384 links between sources) and sub-graphs focused on some entities may be very important (46 links between sources providing the Gene entity).

## 2.3 BioGuide and BBWrapBG: Querying with BioBrowsing

*Expressing a query.* The BioBrowsing query engine is BioGuide, a path-based approach described in [3]. BioGuide provides a generic framework which has

**Table 1.** Graph Features

| *Profile* | *TotNBE* | *TotNBS* | *TotLE* | *TotLS* | *MinL − MaxL(Ns/e)* | *Mean(Ns/e)* |
|---|---|---|---|---|---|---|
| Complete | 10 | 34 | 89 | 838 | 2 - 46 | 9 |
| Medical | 5 | 23 | 25 | 384 | 2 - 46 | 15 |
| Annotation | 6 | 26 | 35 | 485 | 2 - 46 | 13 |



- (1) omim_Disease --CrossRef--> gene_Gene [View in ENTREZ]

**Sort Value : 6.2**

- (2) omim_Disease --CrossRef--> protein_Protein --CrossRef--> gene_Gene [View in ENTREZ]

**Sort Value : 5.6**

- (3) omim_Disease --CrossRef--> pubmed_Biblio --CrossRef--> gene_Gene [View in ENTREZ]
- (4) omim_Disease --CrossRef--> books_Biblio --CrossRef--> gene_Gene [View in ENTREZ]
- (5) omim_Disease --CrossRef--> pmc_Biblio --CrossRef--> gene_Gene [View in ENTREZ]

**Sort Value : 5.2**

- (6) omim_Disease --CrossRef--> nucgss_Nucleotide --CrossRef--> gene_Gene [View in ENTREZ]

3060[uid] OR 3061[uid] OR 3062[uid] OR 3119[uid]

Items 1 - 10 of 10 One page

1: Hcrt Links
**Official Symbol** Hcrt and **Name:** hypocretin [*Rattus norvegicus*]
**Other Aliases:** orexin-A
**Other Designations:** Hypocretin (orexin) neuropeptide
**Chromosome:** 10; **Location:** 10q32.1
**Annotation:** Chromosome 10, NC_005109.2 (89699631..89700848, complement)
**GeneID:** 25723

2: Hcrtr2 Links
**Official Symbol** Hcrtr2 and **Name:** hypocretin (orexin) receptor 2 [*Rattus norvegicus*]
**Other Designations:** hypocretin receptor 2
**Chromosome:** 8; **Location:** 8q24
**Annotation:** Chromosome 8, NC_005107.2 (81068303..81196608, complement)
**GeneID:** 25605

3: Hcrtr1 Links
**Official Symbol** Hcrtr1 and **Name:** hypocretin (orexin) receptor 1 [*Rattus norvegicus*]
**Other Aliases:** Hctr1
**Other Designations:** hypocretin receptor 1
**Chromosome:** 5; **Location:** 5q36
**Annotation:** Chromosome 5, NC_005104.2 (149160739..149170738, complement)
**GeneID:** 25593

4: CPT1B Order cDNA clone, Links
**Official Symbol** CPT1B and **Name:** carnitine palmitoyltransferase 1B (muscle) [*Homo sapiens*]
**Other Aliases:** CPT1-M, KIAA1670, M-CPT1, MCCPT1
**Other Designations:** CPTI, muscle; carnitine O-palmitoyltransferase I, mitochondrial muscle isoform; carnitine palmitoyltransferase 1A (muscle); carnitine palmitoyltransferase 1B
**Chromosome:** 22; **Location:** 22q13.33
**Annotation:** Chromosome 22, NC_000022.9 (49354156..49363862, complement)
**MIM:** 601987
**GeneID:** 1375

**Fig. 2.** BioBrowsing querying process: (top-left) querying interface with the graph of entities and part of the graph of sources in which only sources providing genes are represented; (bottom-left) List of paths between sources generated; (right) answers obtained by clicking on on the link [View in Entrez] next to path (2)

been directly used by BioBrowsing (no changes were necessary in the source code of BioGuide). BioGuide takes as input the two graphs and the user query composed of (i) selected entities in the graph of entities with associated keywords, and (ii) (possibly) sources selected in the graph of sources when the user wants to specify some sources which have to be queried. As an example, the query of Tom is expressed by selecting entities Disease and Gene and specifying `Narcolepsy` as keyword. Tom might restrict the search on sources providing Genes from the (Entrez)Gene source (see Figure 2, top-left).

*Advanced features.* Additionally to the two graphs, BioGuide is able to take into account the preferences the user has on the sources (e.g., users can express that some sources are more reliable than others). BioGuide also offers the possibility of expressing filters on the sources to be queried and on the way they are queried (querying strategy). Examples of filters include specifying that only paths having a maximal length (to be specified) should be followed. Examples of strategies include specifying that sources should be considered in a given order or that additional entities can be considered. In Tom's query, an order has been defined, from sources providing information on Disease to sources providing genes (as a result selected entities are numbered in Figure 2), and additional entities have been considered (e.g., Protein or Biblio). Preference values, filters and strategies can be parameterized by advanced users. Default values have been set using feedback from current users. Complete information on the querying process can be found on [3,5] and will be further discussed in the Related Work section.

*Getting results with BBWrapBG: from BioGuide paths to data entries.* As an intermediate result, a list of paths between sources that could be queried (as alternative ways to get information) is generated by BioGuide. These paths specify for each source which entity it provides. Figure 2 (bottom-left) gives the list of paths obtained by Tom's query, including the following two paths:

(1) $OMIM\_Disease \rightarrow gene\_Gene$ and
(3) $OMIM\_Disease \rightarrow protein\_Protein \rightarrow gene\_Gene$.

The set of paths has been ranked by BioGuide according to the reliability of the sources involved in each path (as introduced above). It allows the user to inspect paths by considering first paths of most interest to him.

When the user selects one of these paths, the BBWrapBG module is run to translate the path into a set of calls to services understood by the Entrez engine to finally provide a set of data entries from Entrez (Figure 2, right-hand side). This translation is not straightforward and involves several intricate calls to Entrez eUtilities.

## 3 Related Work

A plethora of approaches has been proposed in the last twenty years to integrate biological data using various techniques and architectures. According to a recent SIGMOD tutorial [6], current challenges in integrating biological data include providing a *usable system* with (among others) (i) a *presentation data model* to be queried, (ii) a mapping between this layer and the (volatile) schema of the sources to be maintained, and (iii) data results provided to the user in a familiar environment.

Path-based systems [5] have been introduced as loose integration systems which rely on two layers. First, they consider a physical layer, the graph of sources, formed by the sources and their cross-references. Second, they consider a logical layer, the graph of entities, composed of the biological entities and relationships between them. Path-based systems provide alternative paths between sources as alternative ways to get answers and rank paths according to user's preferences. Their aim is to guide the browsing process. The graph of entities in these systems can be seen as the presentation data model [6], it is thus more than just a user-friendly visual interface since it allows the user to pose queries in terms of biological entities instead of having to understand the structures of the sources.

In [5], several path-based systems have been compared including BioMediator [7], BioNavigation [8], DSS [9], and BioGuide [3] (with BioGuideSRS [10]). Systems differ in the structure of their graphs (e.g., multi-labelled edges), the kind of preferences they consider (e.g., reliability, completeness of the sources), the kind of filter preferences they use, their underlying query language, the querying strategies they allow to follow, and the techniques they use to go from paths to data instances (on-the-fly techniques or data warehousing).

However, the systems previously mentioned have considered the two graphs manually designed by their administrator while BioBrowsing is able to generate and maintain semi-automatically a logical layer on top of the 34 current Entrez sources together with the mapping between this layer and the schema of the sources. As for the techniques used to obtain data instances, BioMediator queries a datawarehouse of semi-private data, DSS is used on top of the HKIS-Amadea (private) platform, BioNavigation queries a warehouse containing part of the Entrez data while BioGuideSRS poses on-the-fly queries to the SRS platform. BioBrowsing is using the same query engine as BioGuideSRS but uses new modules able to translate its paths into queries to Entrez. BioBrowsing provides results through the Entrez interface obtained following a complete on-the-fly process.

## 4 Conclusion

The challenges addressed by the BioBrowsing system that we have introduced in this paper are the following:

– Automatically design a network of 34 real, widely used, biological sources
– Semi-automatically generate a logical layer on top of this huge network of sources
– Consider alternative paths in the maze of sources to get complementary information
– Efficiently get access to biological entries from Entrez without considering any warehousing approach.

Our demonstration will highlight the following features:

*Generating graphs and defining profiles.* We will show how BioBrowsing allows to follow both bottom-up and top-down approaches to generate and design graphs. First (Bottom-up), we will show how the two graphs and new profiles can be generated semi-automatically from information on the sources and their links given by the NCBI together with the *contains* information. Second (Top-down), we will show that the graph of entities may conversely be designed *a priori*, and then the mapping between the two graphs been generated. In this part of the demonstration we will consider several alternative sets of entities to demonstrate the flexibility of our approach.

*Querying.* We will consider several biomedical queries to emphasize how (i) providing alternative ways of obtaining results gives access to new information, (ii) knowing that several paths between sources return the same data may augment the confidence the user has in the obtained result, (iii) taking into account preferences in sources allows the user to choose between conflicting information.

As for future work, BioGuideSRS has been very recently used as a testbed to learn preferences users have on sources, based on feedbacks users gave on the data provided by BioGuide paths [11]. BioBrowsing will provide a wider testbed for this kind of work since it will consider much more sources and links. Ongoing work on BioBrowsing also include computing and analyzing the differences between the sets of data obtained by the various paths, following approaches in the spirit of [12]. Our aim in this work is to support the steps which go beyond the browsing phase in the integration process.

## Acknowledgments

This work is supported by the French ANR project MicroBiogenomics. Authors would like to thank the members of the NCBI staff for their guidance in the use of eUtilities. Authors are also very grateful to the reviewers for their valuable comments.

## References

1. Galperin, M.Y.: The molecular biology database collection: 2008 update. Nucleic Acids Research 36, D2–D4 (2008)
2. Zdobnov, E.M., Lopez, R., Apweiler, R., Etzold, T.: The ebi srs server - recent developements. Bioinformatics 18(2), 368–373 (2002)
3. Cohen-Boulakia, S., Davidson, S., Froidevaux, C.: A user-centric framework for accessing biological sources and tools. In: Ludäscher, B., Raschid, L. (eds.) DILS 2005. LNCS (LNBI), vol. 3615, pp. 3–18. Springer, Heidelberg (2005)
4. Nandi, A., Jagadish, H.V.: Assisted querying using instant-response interfaces. In: SIGMOD Conference, pp. 1156–1158 (2007)
5. Cohen-Boulakia, S., Davidson, S.B., Froidevaux, C., Lacroix, Z., Vidal, M.: Path-based systems to guide scientists in the maze of biological data sources. J. Bioinformatics and Computational Biology 4(5), 1069–1096 (2006)
6. Jagadish, H.V., Chapman, A., Elkiss, A., Jayapandian, M., Li, Y., Nandi, A., Yu, C.: Making database systems usable. In: SIGMOD Conference, pp. 13–24 (2007)
7. Shaker, R., Mork, P., Brockenbrough, J.S., Donelson, L., Tarczy-Hornoch, P.: The biomediator system as a tool for integrating biologic databases on the web. In: Proceedings of the Workshop on Information Integration on the Web (held in conjunction with VLDB 2004, ePublication (2004)
8. Lacroix, Z., Morris, T., Parekh, K.: R., L., Vidal, M.E.: Exploiting multiple paths to express scientific queries. In: Scientific and Statistical Database Management (SSDBM), pp. 357–360. IEEE Computer Society, Los Alamitos (2004)

9. Cohen-Boulakia, S., Lair, S., Stransky, N., Graziani, S., Radvanyi, F., Barillot, E., Froidevaux, C.: Selecting biomedical data sources according to user preferences. Bioinformatics 20, i86–i93 (2004)
10. Cohen-Boulakia, S., Biton, O., Davidson, S.B., Froidevaux, C.: Bioguidesrs: querying multiple sources with a user-centric perspective. Bioinformatics 23(10), 1301–1303 (2007)
11. Talukdar, P.P., Jacob, M., Mehmood, M.S., Crammer, K., Ives, Z.G., Pereira, F., Guha, S.: Learning to create data-integrating queries. In: VLDB 2008 (2008)
12. Bao, Z., Cohen-Boulakia, S., Davidson, S.B., Eyal, A., Khanna, S.: Differencing provenance in scientific workflows. In: ICDE (to appear, 2010)

# Using Workflow Medleys to Streamline Exploratory Tasks

Emanuele Santos[1,2], David Koop[1,2], Huy T. Vo[1,2], Erik W. Anderson[1,2], Juliana Freire[2], and Cláudio Silva[1,2]

[1] Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT, USA
[2] School of Computing, University of Utah, Salt Lake City, UT, USA
{esantos,dakoop,hvo,eranders,juliana,csilva}@cs.utah.edu

**Abstract.** To analyze and understand the growing wealth of scientific data, complex workflows need to be assembled, often requiring the combination of loosely-coupled resources, specialized libraries, distributed computing infrastructure, and Web services. However, constructing these workflows is a non-trivial task, especially for users who do not have programming expertise. This problem is compounded for exploratory tasks, where the workflows need to be iteratively refined. In this paper, we introduce *workflow medleys*, a new approach for manipulating collections of workflows. We propose a workflow manipulation language that includes operations that are common in exploratory tasks and present a visual interface designed for this language. We briefly discuss how medleys have been applied in two (real) applications.

## 1 Introduction

The trend towards service-oriented architectures has expanded to a number of domains. Recently, a new class of tools have emerged that help users to leverage and integrate services in a collaborative fashion. Yahoo! Pipes [1] is an example of mashup builder that provides a graphical user interface for assembling pipelines that combine RSS feeds and Web services. Scientific workflow systems, such as Taverna [2] and VisTrails [3], provide a more comprehensive framework which, in addition to services, also supports the integration of general tools and libraries.

The ability to construct complex applications, be they scientific workflows or Web mashups, by weaving services together is very appealing and has many benefits. Although workflow systems are natural candidates as solutions to this problem, there are two important challenges that need to be addressed: usability and support for exploratory tasks. While there has been substantial work on workflow and application integration systems [4], such systems have primarily been designed for power users in enterprise settings. Scientists who use scientific workflow systems do not necessarily have programming expertise. Thus, it is not reasonable to assume that they can write complex control-flow specifications (*e.g.,* using languages such as BPEL [5]), even if a visual programming interface is available.

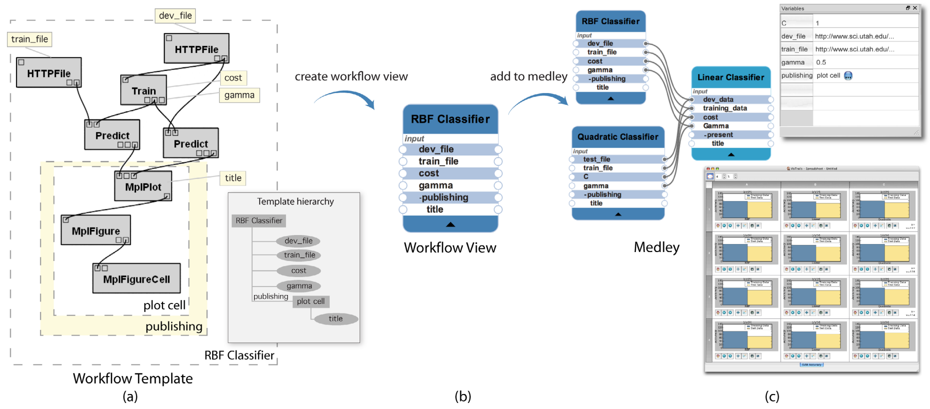
In addition, workflow systems have been traditionally used to automate (complex) processes, which often require a laborious, time-consuming design cycle. In a number of new applications, however, workflows are assembled for exploratory, and sometimes one-of-a-kind tasks. Instead of designing a single workflow that will be run thousands of times, a user (or set of users) manipulates ensembles of workflows that are iteratively refined as she formulates and test hypotheses [6]. Such tasks may require, for example, experimenting with different combinations of parameter values, data sets, or algorithms. Consider the following example of an exploratory task.

**Example 1 (Select best learning classifier) .** To build an effective learning classifier, a user must often tediously build and compare alternative learning techniques as well as experiment with different configurations for each technique. For text classification, support vector machines (SVMs) can be extremely effective [7], but their accuracy depends on a variety of model parameters including the kernel function $f$, the scaling factor $\gamma$, and the penalty parameter of the error term $C$. Different kernel types (linear, polynomial, radial basis function (RBF), and sigmoid) need to be investigated and parameters tuned for each. Thus, selecting the best classifier requires constructing several classifiers, testing them all, and comparing their error rates. Figure 1(c) shows the accuracy rates of three distinct SVM classifiers using C values 0.25, 1, 2, 4 over the test data.

Suppose a workflow designer constructs three workflows, one for each kernel type. Figure 1(a) shows the structure of one of these workflows. This workflow constructs a classifier using training data retrieved from the Web and computes its error rate. Using a similar process, it also derives the error rate for the classifier on the test data. The error rates from the two runs are then sent to a Matplotlib [8] module which generates the plots that are subsequently displayed on the screen. Using a visual programming interface (such as the ones provided by workflow systems [2,9]), to compare the different configurations for $C$ values 0.25, 1, 2, 4, the user has to modify, run and save the results of each workflow. This scenario requires 12 modifications and 12 saved files. Furthermore, if new test (or training) data is made available, the whole process needs to be repeated.

**Workflow Medleys.** In this paper, we propose a new approach to support exploratory tasks that involve ensembles of workflows, or *workflow medleys*. This approach relies on simplified views of workflows that are more intuitive for users, along with operators for manipulating the workflows as a set.

For the scenario above, we desire to create a simplified interface for the given workflows. Note that even small workflows, like the one shown in Figure 1(a) can have many different modules, connections, and parameters, but for most tasks, only a small subset needs to be manipulated. As illustrated in Figure 1(b), simpler views of these workflows might hide all but this subset of entities. In our example, these might include the names of the input files (`dev_file` and `train_file`) and the $C$ parameter (`cost`). Then, to compare the different classifier configurations, as Figure 1(c) illustrates, we can *synchronize* the appropriate parameters across workflows, and then set all of these parameters to the appropriate values. Note that setting the value of a single parameter, *e.g., cost*, updates all parameters that are synchronized with it. This provides a means to efficiently compute and compare the 12 different methods. Also, if new test data is made available, all plots can be re-generated by updating a the `dev_file` parameter only once.



Variables

| | |
|---|---|
| C | 1 |
| dev_file | http://www.sci.utah.edu/... |
| train_file | http://www.sci.utah.edu/... |
| gamma | 0.5 |
| publishing | plot cell |

**Fig. 1.** Example of creating a medley for Example 1. (a) The developer marks configurable pieces of the workflow to create a template. (b) A workflow view is created based on the workflow template. (c) A medley of SVM Classifiers. The connections mean that synchronization is taking place.

Note that the same mechanism used to explore parameter spaces can be used to update the workflow definitions in bulk. For example, by synchronizing the subworkflow (variable `publishing` in the medley shown in Figure 1(c)) which consists of the modules responsible for displaying the results, a single update operation can be used to replace this subworkflow with a different set of modules. For example, instead of displaying the results on the screen, the new version may generate an HTML page with the images embedded.

**Outline and Contributions.** We propose a new approach that streamlines exploratory tasks that require the composition of multiple workflows. This approach is general and can be combined with existing workflow and workflow-based systems, such as Yahoo! Pipes, Taverna and Kepler. In addition, it can be naturally mapped into an intuitive interface that is suitable for users that are not expert programmers. We introduce the medley model in Section 2. This model consists of a set of concepts and operations for manipulating workflows and captures operations that are common in exploratory tasks. In Section 3 we discuss our first prototype of a user interface for the medley model. We describe how it is implemented and used. We have explored the use of medleys in two real applications: chemical informatics and a comparative analysis of isosurface extraction algorithms. We describe our experiences in Section 4. In Section 5, we review related work and we conclude in Section 6.

## 2 Manipulating Workflow Specifications

In this section, we show how manipulation of a workflow collection can be simplified by developing the concepts of workflow templates, workflow views, and medleys. We begin by reviewing the definition of a workflow along with basic workflow operations, and then introduce the *workflow template* as a way for designers to designate configurable pieces of a workflows. A *workflow view* is the projection of a workflow according to a workflow template, and a *medley* is a collection of workflow views along with a set of links between them that synchronize or compose the views. Throughout this section, we assume a dataflow model for workflows [10]. Note that all of the operations and concepts we introduce are independent of the underlying workflow management system.

## 2.1 Workflows

**Definition 1.** A *workflow* $w(M, C)$ is a set of *modules*, $M$, along with a set of *connections*, $C$, linking the modules. Each module $m \in M$ is associated with a tuple $(I_m, O_m, P_m)$, where $I_m$ corresponds to a set of *input ports*, $O_m$ corresponds to a set of *output ports*, and $P_m$ is a list of *parameters*. Each parameter $p \in P_m$ is associated with a value $v$. A *connection* $(o, i)$ links an output port $o$ from a module $m_1$ to an input port $i$ of another module $m_2$. $o \in O_{m_1}$ is the *source port* and $i \in I_{m_2}$ is the *target port*. $m_1$ and $m_2$ can only be connected through ports $o$ and $i$ if the *types* of the ports are compatible. *Sources* are modules where no target port is connected, and *sinks* are modules whose no source port is connected. Parameters can also have a type, and the value of a parameter must be an instance of that type. ■

**Definition 2.** Given a workflow $w(M, C)$, a *subworkflow* $w_s(M', C')$ is a workflow where $M' \subset M$ and $C' \subset C$ such that $c \in C'$ if and only if $c$ connects $m_1$ to $m_2$ where $m_1, m_2 \in M'$. ■

While there are a variety of workflow operations we could discuss, we will highlight two: *enactment*, executing a workflow, and *substitution*, changing workflow components. These operations can be used both when designing a workflow and when interacting with a completed workflow.

**Enactment.** A *workflow enactment* is the execution of a workflow in the order determined by the network of modules and connections. We recursively update modules starting with the sinks until all modules are "up-to-date". Because each module depends on all of its inputs, these data requests propagate all the way to the sources (which reference the initial data), who update their outputs, allowing modules connected to their output ports to then execute. Execution continues until each sink has been executed.

**Substitution.** *Substitution* allows workflow components (*e.g.,* parameter values and modules) to be replaced. More formally, given a workflow $w(M, C)$, the operation *substituteParameter*$_w(m, p, v)$ assigns value $v$ to a parameter $p$ of a module $m$ in workflow $w$, provided that the types of $v$ and $p$ are compatible. Given a second workflow $w_s(M', C')$, the operation *substituteWorkflow*$_w(M^*, w_s)$ replaces the subworkflow induced by the modules in $M^* \subset M$ with the workflow $w_s$. In order to accomplish this, we must create the connections that link $w_s$ back into the workflow $w$. Each connection that links the modules in $M^*$ to those in $M - M^*$ is remapped to a connection linking $M'$ to those in $M - M^*$ by matching the types of the ports in the original connections to match those in the new connections.

## 2.2 Simplifying Workflows

As outlined earlier, workflow systems allow users to create and execute workflows. A limitation of these systems is the difficulty involved in modifying an existing workflow by users other than the original workflow developer. Our goal is to simplify these modifications and allow users to interact with ensembles of workflows in a more intuitive manner. Our approach is to allow the designer to designate configurable pieces of the workflow through workflow templates. Such designations help users determine proper inputs as well as experiment with different workflow variations. From such templates, we can create workflow views that abstract much of the complexity of workflows. Users can then combine workflow views in medleys using synchronization and composition operations.

**Workflow Templates.** We introduce a *workflow template* as a workflow that allows designers to define reconfigurable pieces of the workflow in a hierarchical way. Users can select and label parameters or subworkflows using a nomenclature that is meaningful for a given application or task. Figure 1(a) shows a workflow template generated for the classification workflow described in Example 1.

Note that the designer selected a subset of parameters as well as the plotting subworkflow that should be exposed. The root of the template hierarchy represents the workflow, and its children and descendants correspond to configurable parameters and subworkflows. We refer to each element in the the template hierarchy as a *workflow template node*. Nodes that correspond to subworkflows are represented as rectangles and parameters as ellipses. Note that labels are unique in a given hierarchy level. By representing the template as a hierarchy, our approach is able to handle arbitrary nesting of workflows.

Workflow template nodes provide the same operations of a workflow as well as other specific operations for labeling and removing labels, for creating, adding and removing child nodes, creating and removing connections between template nodes and between template nodes and modules, and for materializing a workflow.

**Workflow Views.** In a workflow template, important and configurable elements (*i.e.,* modules, parameters, and subworkflows) are selected, and a workflow view effectively hides all unselected elements. More formally, a *workflow view* $w_v$ is a projection of a workflow $w$ where only a subset of the workflow elements are exposed for direct interaction. We refer to the exposed elements as *variables*. Any workflow element not exposed by a workflow view cannot be directly changed in the view. However, a workflow view maintains a reference to the original workflow, and thus views can be enacted by enacting the underlying workflow. Notice that a workflow view can be generated from a workflow template. In fact, the parameters and configurable subworkflows are also represented as a hierarchy that mirrors the one for the template hierarchy. Figure 1(b) shows a view (`RBF Classifier`) derived from the template in Figure 1(a).

**Medleys.** For exploratory tasks, a user often needs to create and manipulate a set of workflows, as shown in our machine learning example. To support this, we introduce a *medley* $\mathbb{M}$ as a collection of (related) workflow views along with a set of relationships between the views. These relationships are defined by operations linking the views, including synchronization and composition.

When two views are *synchronized*, one or more variables from each view are linked. A variable $x$ in a workflow view $w_v \in \mathbf{M}$ can be synchronized with any variable $x'$ in another view $w'_v \in \mathbf{M}$ if $x$ and $x'$ have the same type. Then, for any pair of linked variables, binding either to a value $v$ ensures that each variable is set to $v$.

The ability to synchronize variables is useful for tasks like comparative visualization since we can ensure that parameters across different workflows whose values should be the same will indeed be the same. Consider again the machine learning example, and suppose we have a medley with views for the workflows that use the different classifiers. By synchronizing their input files and cost values, a user could quickly set these parameters once and their values would be automatically propagated to the three workflows. Furthermore, synchronization enables a user to efficiently explore different configurations. Instead of setting values for each workflow individually—which can be both time consuming and error prone, the value for a parameter is set only once and is automatically propagated to all synchronized variables in multiple views.

Two views are *composed* by connecting an output port in one view to the input port of the other. In our example, composition could be used to pass the HTML file generated by the two classifier views to a view that sends files to a web server via FTP. In addition, a medley can combine composition and synchronization to easily construct a variety of analyses and explorations.

Note that we could consider synchronization or composition on workflows instead of workflow views, but this could be much more complicated for the user. Because workflow views reduce the number of components that are exposed, they make it much easier to identify how workflows can be integrated and synchronized.

## 3 Creating and Interacting with Medleys

While workflow templates, workflow views, and medleys allow users to simplify and integrate workflows, constructing these concepts needs to be straightforward. For this reason, we have implemented these operations using an intuitive user interface. In this section, we describe our initial implementation of such an interface.

**Creating Workflow Templates and Views.** Developers use the *Workflow Template Editor* to create a workflow template, by selecting and labeling parameters and subworkflows, as shown in Figure 1(a). Given a workflow template, displaying the corresponding workflow view requires a simplified interface. In our implementation, we use a table-based layout where each variable name and editable value are displayed (see Figure 1(c)).

Once a template is created, one of the operations supported by the Template Editor is view creation. While configuring a view, users can set the visibility of the parameters and configurable subworkflows, as well as select suggestions from the list stored in the template. These suggestions will guide the end users to pick meaningful values for the parameters when they are not familiar with the workflows. Note that both templates and views can be stored in a repository where they can be accessed later.

**Creating and Manipulating Medleys.** To combine workflow views in a medley, the developer uses the *Medley Editor*. The views stored in the Workflow View Repository are displayed on a panel and they can be dragged and dropped on a canvas. Once on the canvas, the medley operations (*i.e.,* synchronization and composition) can be applied to the views. A screenshot of part of the Medley Editor is shown in Figure 1(c).

Each variable in a view has an associated handle (see the circles on the left and right of each variable name in the workflow view in Figure 1(b)). By connecting the handles for two variables in two distinct views, their values are synchronized. To simplify the task of identifying variables to be synchronized, when the developer starts to create a connection all the variables that are compatible with that variable are highlighted.

**Demonstration Overview.** In this demonstration, we will use this interface to create and manipulate medleys for exploratory tasks in scientific visualization scenarios. In particular, we will demonstrate how to create workflow views and how to synchronize their variables in a medley.

## 4 Case studies

We tested how medleys can be used in exploratory tasks in two different applications.

**Integrating Chemical Informatics Web Services.** The first application consisted of integrating chemical informatics web services to locate information about a specific compound and graphically visualize it. To perform this task, a user must invoke several services provided by Chembiogrid [11]. The first workflow fetches the SMILES [1] code of a molecule id. The second and third workflows fetch the 2D image and the 3D model representing the SMILES code, respectively. As the user is not able to render the 3D model in the format returned by the web service, another web service, *sdfToPdb*, is used to convert the data to pdb format. Finally, a fourth workflow is used to render and display the molecule using a ball-and-stick model. Completing this task using a workflow system that supports Web services, such as for example, Taverna, a user needs to assemble a workflow that combines these four workflows, carefully connecting outputs to inputs. In contrast, by creating a medley with workflow views created for the four workflows described above, the user can synchronize and compose the workflows without having to directly modify the structure of workflows.

**Comparative Analysis of Isosurface Extraction Algorithms.** One of our collaborators needed to perform a comparative analysis of several algorithms for extracting isosurfaces [12], involving the visualization of the meshes produced by the different algorithms and the histograms that accumulate quality information on each mesh. Although a workflow system would help him structure his experiment (*e.g.,* by creating a workflow for each algorithm that both renders the mesh and displays the histogram), it would require him to modify the parameters on each workflow one by one, and repeat this tedious process over and over until a good visualization is found. We created a medley containing workflow views for each algorithm our collaborator wanted to evaluate. He performed the comparative analysis using this medley, by changing parameters and

[1] SMILES stands for Simplified Molecular Input Line Entry Specification and it is a linear notation that uses alphanumeric characters to encode molecular structure.

datasets and without having to manipulate the workflows directly. Although the use of a workflow system would help on structuring the experiments, it would still require users to modify the parameters on each workflow one by one, and repeat this tedious process over and over until a good result is found.

## 5 Related Work

Workflows and workflow-based systems have emerged as an alternative to ad-hoc approaches to data exploration commonly used in the scientific community [9,2,13,3,14,15,16]. Workflow systems provide languages with well-defined semantics to specify computational processes which integrate existing applications according to a set of rules [4,17,18,19]. Not only do they support the automation of repetitive tasks, but they can also capture complex analysis processes at various levels of detail and systematically capture provenance information for the derived data products [20]. Workflow systems, however, have been primarily designed for expert programmers and to automate repetitive processes. Our goal with the medley approach is to provide casual users with intuitive interfaces to combine services on-the-fly and perform exploratory tasks through workflows.

Social Web sites and web-based communities (*e.g.,* Flickr [21], Facebook [22], Yahoo! Pipes [1]), which facilitate collaboration and sharing between users, are becoming increasingly popular. An important benefit of these sites is that they enable users to leverage the *wisdom of the crowds*. In the (very) recent past, a new class of Web site has emerged that enables users to upload and collectively analyze many types of data (see *e.g.,* [23,24]). These are part of a broad phenomenon that has been called "social data analysis" [25]. Many Eyes [23] (developed at IBM research) is a site for sharing and commenting on visualizations. Users can upload any data set and visualize the data using a wide range of tools provided by Many Eyes (*e.g.,* line graphs, stack graphs, bar charts, block histograms, treemaps). This trend is expanding to the scientific domain where there is an increasing number of collaboratories. An example of a social Web community in this domain is the new myExperiment site [26]. Their goal is to enable "scientists to share, re-use and re-purpose their workflows and reduce time-to-experiment, share expertise and avoid reinvention". The medley infrastructure can be integrated with these sites to provide a flexible mechanism for users to combine multiple workflows and services from a large, shared pool. In such a scenario, medleys can also serve as an unobtrusive mechanism for capturing semantics and domain-specific knowledge. For example, when a user synchronizes components from different services, this indicates that these components are related (and compatible). Such knowledge can be re-used to help other users compose new applications.

Biton et al. [27] proposed the creation of views over workflows. Their views are similar to our notion of workflow view. Their objective, however differs from ours in that their goal is to deal with the overload of provenance derived from the workflow runs, by controlling the granularity at which provenance is collected (or published) through these views.

## 6 Conclusion

Workflow medleys represent a new approach for manipulating ensembles of workflows. Our framework combines a set of operations that are common in exploratory tasks with an intuitive visual interface. We have studied our approach by examining ways it could be applied to different application areas, and have seen that medleys help simplify workflow-based exploratory tasks. We plan to conduct user studies to further evaluate our approach with respect to usability and effectiveness.

**Acknowledgments.** This work is partially supported by the NSF (under grants IIS-0513692, CCF-0401498, EIA-0323604, CNS-0514485, IIS-0534628, CNS-0528201, OISE-0405402), the DOE, and an IBM Faculty Award. E. Santos is partially supported by a CAPES/Fulbright fellowship.

## References

1. Yahoo! Pipes, `http://pipes.yahoo.com`
2. The Taverna Project, `http://taverna.sourceforge.net`
3. The VisTrails Project, `http://www.vistrails.org`
4. Aalst, W., Hee, K.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge (2002)
5. Business process execution language for web services version 1.1 (February 2008), `http://www.ibm.com/developerworks/library/specification/ws-bpel`
6. Freire, J., Silva, C.T., Callahan, S.P., Santos, E., Scheidegger, C.E., Vo, H.T.: Managing rapidly-evolving scientific workflows. In: Moreau, L., Foster, I. (eds.) IPAW 2006. LNCS, vol. 4145, pp. 10–18. Springer, Heidelberg (2006)
7. Mitchell, T.: Machine Learning. McGraw-Hill, New York (1997)
8. The matplotlib library, `http://matplotlib.sourceforge.net`
9. The Kepler Project, `http://kepler-project.org`
10. Lee, E.A., Parks, T.M.: Dataflow Process Networks. Proceedings of the IEEE 83(5), 773–801 (1995)
11. The Chembiogrid web site, `http://www.chembiogrid.org`
12. Schroeder, W., Martin, K., Lorensen, B.: The Visualization Toolkit An Object-Oriented Approach To 3D Graphics. Kitware (2003)
13. Parker, S.G., Johnson, C.R.: SCIRun: a scientific programming environment for computational steering. In: Supercomputing (1995)
14. Deelman, E., Singh, G., Su, M.H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A., Jacob, J.C., Katz, D.S.: Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. Scientific Programming Journal 13(3), 219–237 (2005)
15. Microsoft Workflow Foundation, `http://msdn2.microsoft.com/en-us/netframework/aa663322.aspx`
16. Foster, I., Voeckler, J., Wilde, M., Zhao, Y.: Chimera: A virtual data system for representing, querying and automating data derivation. In: Statistical and Scientific Database Management (SSDBM), pp. 37–46 (2002)
17. Lawrence, P. (ed.): Workflow Handbook. Workflow Management Coalition. John Wiley and Sons, Chichester (1997)

18. van der Aalst, W.: Business process management: A personal view. Business Process Management Journal 10(2), 135–139 (2004)
19. Mohan, C., Alonso, G., Günthör, R., Kamath, M.: Exotica: A research perspective of workflow management systems. IEEE Data Engineering Bulletin 18(1), 19–26 (1995)
20. Deelman, E., Gil, Y.: NSF Workshop on Challenges of Scientific Workflows. Technical report, NSF (2006), `http://vtcpc.isi.edu/wiki/index.php/Main_Page`
21. Flickr, `http://www.flickr.com`
22. Facebook, `http://www.facebook.com`
23. Viegas, F.B., Wattenberg, M., van Ham, F., Kriss, J., McKeon, M.: Many eyes: A site for visualization at internet scale. IEEE Transactions on Visualization and Computer Graphics 13(6), 1121–1128 (2007)
24. Swivel, `http://www.swivel.com`
25. Social data analysis workshop (2008) , `http://researchweb.watson.ibm.com/visual/social_data_analysis_workshop`
26. Myexperiment, `http://www.myexperiment.org`
27. Biton, O., Cohen-Boulakia, S., Davidson, S.B.: Zoom*userviews: querying relevant provenance in workflow systems. In: VLDB 2007: Proceedings of the 33rd international conference on Very large data bases, VLDB Endowment, pp. 1366–1369 (2007)

# Experiences on Processing Spatial Data with MapReduce[*]

Ariel Cary, Zhengguo Sun, Vagelis Hristidis, and Naphtali Rishe

Florida International University
School of Computing and Information Sciences
11200 SW 8th St, Miami, FL 33199
{acary001,sunz,vagelis,rishen}@cis.fiu.edu

**Abstract.** The amount of information in spatial databases is growing as more data is made available. Spatial databases mainly store two types of data: raster data (satellite/aerial digital images), and vector data (points, lines, polygons). The complexity and nature of spatial databases makes them ideal for applying parallel processing. MapReduce is an emerging massively parallel computing model, proposed by Google. In this work, we present our experiences in applying the MapReduce model to solve two important spatial problems: (a) bulk-construction of R-Trees and (b) aerial image quality computation, which involve vector and raster data, respectively. We present our results on the scalability of MapReduce, and the effect of parallelism on the quality of the results. Our algorithms were executed on a Google&IBM cluster, which became available to us through an NSF-supported program. The cluster supports the Hadoop framework – an open source implementation of MapReduce. Our results confirm the excellent scalability of the MapReduce framework in processing parallelizable problems.

## 1 Introduction

Geographic Information Systems (GIS) deal with complex and large amounts of spatial data of mainly two categories: raster data (satellite/aerial digital images), and vector data (points, lines, polygons). This type of data is periodically generated via specialized sensors, satellites or aircraft-mounted cameras (sampling geographical regions into digital images), or GPS devices (generating geo-location information). GIS systems have to efficiently manage repositories of spatial data for various purposes, such as spatial searches, and imagery processing. Due to the large size of spatial repositories and the complexity of the applications to process them, traditional sequential computing models may take excessive time to complete. Emerging parallel computing models, such as MapReduce, provide a potential for scaling data processing in spatial applications.

The goal of this paper is to present to the research community our experiences from using the MapReduce model to tackle two typical and representative spatial data

[*] This research was supported in part by NSF grants IIS-0837716, CNS-0821345, HRD-0833093, EIA-0220562, IIS-0811922, IIP-0829576 and IIS-0534530, and equipment support by Google and IBM.

processing problems. The first problem involves vector spatial data and the second involves raster data.

The first problem is the bulk-construction of R-Trees [1], a popular indexing mechanism for spatial search query processing. We show how previous ideas, like the ordering of multi-dimensional objects via space-filling curves, can be used to create a MapReduce algorithm for this problem. We also discuss how our solution is different from previous approaches on parallelizing the construction of an R-Tree.

The second problem processes aerial digital imagery, and computes and stores image quality characteristics as metadata. Original images may contain inaccurate, distorted, or incomplete data introduced at some phase of imagery generation; for example, a portion of an image may be completely blank. Pre-computed metadata is important in dynamic imagery consistency checking, and allows the appropriate mosaicing with better sources to improve the imagery display. This problem is naturally parallelizable since each tile can be potentially processed independently. In practice, the amount of data processed by each cluster processor depends on the file system characteristics like the minimum processing unit.

Both problems were solved and evaluated on a Google&IBM cluster supplied by the NSF Cluster Exploratory (CluE) program [2][3]. We present our experiences on using such a cluster in practice and deploying MapReduce jobs.

The key contribution of this work is as follow:

- We present techniques for bulk building R-trees using the MapReduce framework.
- We present how MapReduce can be applied to massively parallel processing of raster data.
- We experimentally evaluated our algorithms in terms of execution time, scalability and quality of the output. We provide various metrics to measure the quality of the resulting R-Tree.

This paper is organized as follows. Section 2 describes the steps in deploying MapReduce applications on the Google&IBM's cluster, as well as some physical configurations. Sections 3 and 4 present the detailed MapReduce algorithms for our two target problems. Section 5 presents experimental results of our algorithm implementations for different settings. Section 6 discusses related works. Last, Section 7 concludes our work.

## 2 Using MapReduce in Practice

The cluster used in this paper is provided by the Google and IBM Academic Cluster Computing Initiative [2][3]. The cluster contains around 480 computers (nodes) running open source software including the Linux operating system, XEN hypervisor and Apache's Hadoop [4], which is an open source implementation of the MapReduce programming model. Each node has half terabytes storage capacity summing up to about 240 Terabytes in total. Access to the cluster is provided through the Internet by a SOCKS proxy server. SOCKS is an Internet protocol that secures client-server communications over a non-secure network.

There are three main steps in interacting with the cluster, as shown in Figure 1. (1) Input data is uploaded into the cluster. The user uses file system shell scripts provided

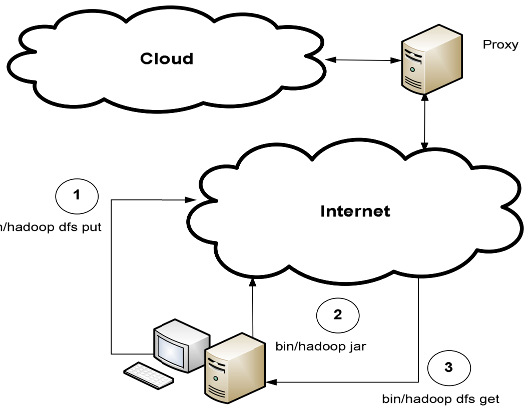


**Fig. 1.** Google, IBM Academic Cluster Overview

by the Hadoop Distributed File System (HDFS), which is an integral part of the Apache Hadoop project; HDFS is a clone project of Google's files system GFS [5]. (2) A user develops a Hadoop application and submits it to the cluster via Hadoop command. Hadoop applications are usually developed in Java, but other languages are supported, like C++ and Python. (3) After application execution is completed, the output is downloaded to the user's local site with Hadoop file system shell scripts.

MapReduce programming model requires expressing the solutions with two functions: map and reduce. A map function takes a key/value pair, executes some computation, and emits a set of intermediate key/value pairs as output. A reduce function merges all intermediate values associated with the same intermediate key, executes some computation on them, and emits the final output. More complex interactions can be achieved by pipelining several MapReduce compounds in a workflow fashion. A data set is stored as a set of files in HDFS, which are in turn stored as a sequence of blocks (typically of 64MB in size) that are replicated on multiple nodes to provide fault-tolerance. An interested reader may refer to MapReduce Google's work [6] and open source Hadoop documentation [4] for a detailed description of MapReduce and Hadoop concepts.

## 3 Building R-Tree with MapReduce

This section discusses a MapReduce-based algorithm for building an R-Tree index structure [1] on a spatial data set in parallel fashion. Let us start our description by defining the problem. Let $D$ be a spatial data set composed of objects $o_i$, $i=1, .., |D|$. Each object $o$ has two attributes <*o.id*, *o.P*>, where *o.id* is the object's unique identifier and *o.P* is the object's location in some spatial domain; other attributes are possible, but we concentrate on these only for our R-Tree construction purpose. The R-Tree minimum bounding rectangles (*MBRs*) are created based on the objects' spatial attribute *o.P*. Identifiers *o.id* are used as references to objects stored in the R-Tree leaves.

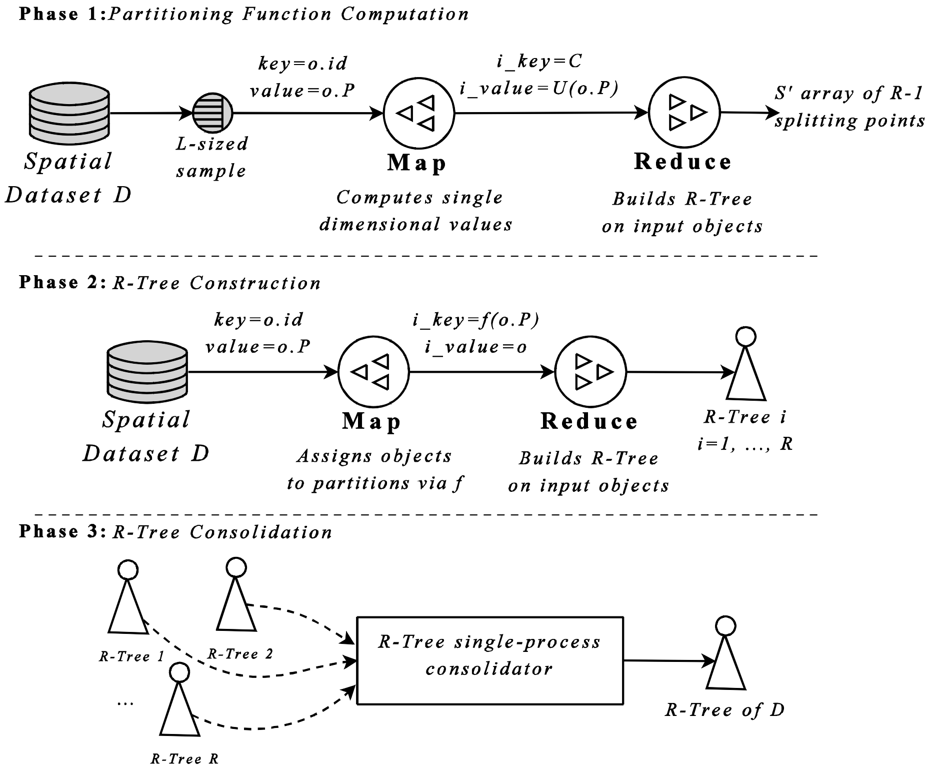

Phase 1: Partitioning Function Computation
key=o.id
value=o.P
i_key=C
i_value=U(o.P)
Spatial Dataset D
L-sized sample
Map
Computes single dimensional values
Reduce
Builds R-Tree on input objects
S' array of R-1 splitting points
Phase 2: R-Tree Construction
key=o.id
value=o.P
i_key=f(o.P)
i_value=o
Spatial Dataset D
Map
Assigns objects to partitions via f
Reduce
Builds R-Tree on input objects
R-Tree i
i=1, ..., R
Phase 3: R-Tree Consolidation
R-Tree 1
R-Tree 2
...
R-Tree R
R-Tree single-process consolidator
R-Tree of D

**Fig. 2.** Phases involved in building an R-Tree index for a data set *D* in MapReduce

The proposed method consists of three phases executed in sequence, as can be seen in Figure 2. First, the spatial objects are partitioned into groups. Then, each group is processed to create a small R-Tree. Finally, the small R-Trees are merged into the final R-Tree. The first two phases are executed in MapReduce, while the last phase does not require high computational power, thus it is executed sequentially outside of the cluster.

The three main phases of the algorithm are:

1 Computation of *partitioning* function *f*. The inputs for this phase are the data set *D* and a positive number *R,* which represents the number of partitions. The purpose of *f* is to assign any object of *D* into one of the *R* possible partitions. The function is computed in such a way that applying *f* on *D* yields *R* (ideally) equally-sized partitions. In practice, minimal variance in sizes is acceptable. At the same time, *f* attempts to put objects that are close in the spatial domain in the same partition. The output of this phase is a function *f* which takes as input an object location *o.P* and outputs a partition number. Note that no actual partitioning or data moving happens at this point. The next phase utilizes *f* for such purpose. More details of this step are presented in Section 3.1.

2 *R-Tree construction*. During this phase, the function *f* calculated in the first phase is used by *Mappers* to divide *D* into *R* partitions. Then, *R Reducers* build *R* independent "small" R-Tree indices simultaneously on their input partitions. The output of this phase is a set of *R* independent R-Trees. Details of this step are presented in Section 3.2.

3 R-Tree *consolidation*. This phase combines the *R* individual R-Trees, built in the second phase, under a single root node to form the final R-Tree index of *D*. This phase can be as simple as making the *R* R-Trees children of a single root node, or it may require adding a few extra levels (at most one in practice) if *R* exceeds the capacity of a single node. Since this phase is not computationally intensive for *R* under a few hundreds or thousands, it is executed by a single process outside the cluster. The logic to run this phase is fairly simple, so no further elaboration will be done on this step.

## 3.1 Partitioning Function

The purpose of the partitioning function *f* is to provide a means for assigning objects of *D* to a pre-defined number of *R* partitions. We use the idea of mapping multi-dimensional spaces into an ordered sequence of single-dimensional values via space-filling curves for this purpose. This idea has been studied in the literature as a way to numbering objects in multi-dimensional spaces [7, 8]. In our present problem, we map objects' location attribute *o.P* into such curves. We use the Z-order curve [9] in our experiments in Section 5.1. The partition number of an object *o* is determined by *f*(*o.P*), which evaluates to a value from the set $\{1, 2, .., R\}$. By using a space-filling curve, the partitioning function *f* achieves two goals:

- Generate *R* (almost) uniformly-sized partitions, and
- Preserve spatial locality. If two distinct objects *o1* and *o2* are close to each other in the spatial domain, then they are likely to be assigned to the same partition, i.e. $f(o1.P) = f(o2.P)$.

Next, we propose a MapReduce algorithm to define *f*.

*MapReduce Algorithm*

The general idea is inspired by the TeraSort Hadoop application [10], which partitions an input da*ta set* via d*ata* sampling. Given a data set *D* and target number of partitions *R*, the MapReduce algorithm runs *M* Mappers that collectively take *L* sample objects from *D* (that is, each Mapper samples $\frac{L}{M}$ objects) and emit their single-dimensional values $S=\{U(o_i.P), i=1, .., L\}$ given a space filling curve *U*. Then, a single Reducer sorts *S*, and determines a list $S'$ of $R-1$ splitting points that split the ordered sequence of samples into *R* equal-sized partitions. Then, in general, an object *o* belongs to partition *j* if $S'[j-1] < U(o.P) \leq S'[j]$. Thus, *f* utilizes the splitting points in $S'$ to assign objects to partitions.

The specific MapReduce key/value input pairs as well as outputs are presented in Table 1. Mappers read in total *L* samples at random offsets of their input *D*, and compute their single dimensional value with the space-filling curve *U*. The intermmediate key equals to *C* which is a constant, whose value is irrelevant, that helps in sending Mappers' outputs to a single Reducer. The Reducer receives the *L* single-dimensional values generated by Mappers, and sorts them into an auxiliary list $u_1, .., u_L$, from which $R$-$1$ elements are taken starting at the $(\frac{L}{R})$-*th* element and subsequently at fixed-length offsets to form a list $S'$ of splitting points.

**Table 1.** Map and Reduce inputs/outputs in computing partitioning function *f*

| Function | Input: (Key, Value) | Output: (Key, Value) |
|---|---|---|
| *Map* | $(o.id, o.P)$ | $(C, U(o.P))$ |
| *Reduce* | $(C, list(u_i, i=1, .., L))$ | $S'$ |

An important observation in the sampling process is that Mappers read input data from the distributed storage at block-sized amounts, which is a Hadoop distributed file system parameter specifically tuned for load balancing large files across storage nodes. Thus, all Mappers, except perhaps for the last one, will read the same amount of data, equal to the file system's block size.

The rationale of the splitting points in $S'$ is that they provide good enough *boundaries* to sub-divide $D$ into $R$ partitions since they come from randomly sampled objects. Experiments in section 5.1 show very low standard deviation (under 1%) on the number of objects per partition. Formally, the function *f* is defined as follows:

$$f(o.P) = \begin{cases} 1, & U(o.P) \leq S'[1] \\ j, & S'[j-1] < U(o.P) \leq S'[j], j = 2, \dots, R-1 \\ R, & U(o.P) > S'[R-1] \end{cases} \tag{1}$$

This computation is characterized by running multiple *Mappers* (samplig data) and one *Reducer* (sorting samples), which may become a limiting factor in scalability. If the size of *S* becomes sufficiently large, then the TeraSort [10] approach can be used to sort its items in parallel, which makes the algorithm more scalable.

## 3.2 R-Tree Construction

In this phase, *R* individual R-Tree indices are built concurrently. Mappers partition the input data set *D* into *R* groups using the partitioning function *f*. Then, every partition is passed to a different Reducer, which independently builds an R-Tree on its input. Next, every Reducer outputs a root node of their constructed R-Trees, so *R* subtrees are written to the file system at the end of this phase.

Input and output key/value pairs are shown in Table 2. Mappers read their input data in its entirety and compute objects assigned partitions via *f*(*o.P*). Then, every Reducer receives a number of input objects *A* for which an R-Tree is built and its root emitted as output. Since *f* balances partitions, it is expected that all Reducers will receive a similar number of objects $(A \sim \frac{|D|}{R})$, thus executing similar amount of work in constructing their R-Trees. However, good balancing depends on the underlying space-filling curve *U* used by *f*, and the number of sampled objects *L*. More samples help in tuning the splitting points, but incur in larger sorting time of *L* elements.

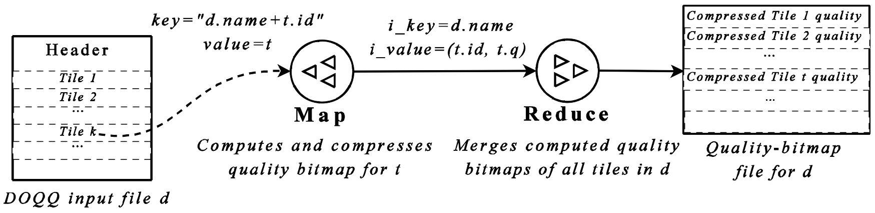
**Table 2.** MapReduce functions in constructing R-Trees

| Function | Input: (Key, Value) | Output: (Key, Value) |
|---|---|---|
| *Map* | (*o.id*, *o.P*) | (*f(o.P)*, *o*) |
| *Reduce* | (*f(o.P)*, *list*($o_{i,\ i=1,\ldots,A}$)) | *tree.root* |

Another concern is the quality of the produced R-Trees in relation to the parameter *R*. In Section 5.1, we provide some initial insight into this direction by measuring R-Tree parameters such as area and overlap in a simplified way, and plotting their *MBRs* for visual analysis.

## 4 Tile Quality Computation Using MapReduce

This section discusses a MapReduce algorithm to compute the quality information of aerial/satellite imagery. Such information is useful for fast identification of defective image portions, e.g. blank regions inside a tile or a group of tiles, and subsequent dynamic image patching using better imagery available at rendering time. For a given tile, we define a pixel as "bad" if all the values of its samples are below or above some predefined value.



key="d.name+t.id"
value=t
Header
Tile 1
Tile 2
...
Tile k
...
DOQQ input file d
i_key=d.name
i_value=(t.id, t.q)
Map
Computes and compresses quality bitmap for t
Reduce
Merges computed quality bitmaps of all tiles in d
Compressed Tile 1 quality
Compressed Tile 2 quality
...
Compressed Tile t quality
...
Quality-bitmap file for d

**Fig. 3.** Tile quality computation algorithm overview

Image tiles are stored in customized DOQQ files [11], augmented with a descriptive header. Let *d* be a DOQQ file and *t* be a tile inside *d*, *d.name* is *d*'s file name and *t.q* is the quality information of tile *t*. More details of our data set are presented in Section 5.2. Figure 3 depicts the execution overview of our MapReduce algorithm. The algorithm runs on a tile by tile basis within the boundaries of a given DOQQ file, computing a bitmap per tile where a tile pixel is associated to a bit that is set to 1 if the pixel is deemed "bad", and 0 otherwise.

*MapReduce Algorithm*

Each DOQQ file is first partitioned into several splits, each of which is then processed by a separate Mapper. Splits are carefully generated by parsing tiles out of the input file until the size of all the tiles is close (little smaller) to the block size of the underlying distributed file system or end of file is reached. In doing so, tile boundaries are preserved between different splits. Then, each Mapper will have to read at most two blocks of a file. This helps reduce data transfer time between nodes because different blocks of a file are usually stored on separate nodes. Tiles (values) inside one split are identified by *d.name* and *t.id* (keys) and combined as key/value input for Mappers.

**Table 3.** Input and output of map and reduce functions

| Function | Input: (Key, Value) | Output: (Key, Value) |
|---|---|---|
| *Map* | (*d.name+t.id*, *t*) | (*d.name*, *t.q*) |
| *Reduce* | (*d.name*, *list*(   *.q*)) | *Quality-bitmap of d* |

The input and output key/value pairs for Mappers and Reducers are described in Table 3. The Mapper decompresses the JPEG tile *t*, iterates through each pixel of *t* to obtain quality information *t.q* (a bitmap, one bit per pixel) and compresses it using Run-length encoding (RLE) algorithm. After that, it emits the intermediate key/value pair with *d.name* as the key and *t.q* as the value. The Reducer merges all the *t.q* bitmaps that belongs to a file *d* and writes them to an output file, containing image quality for *d*, as shown in Figure 3.

## 5 Experiments

This section presents and discusses the experimental results we obtained by running the algorithms described in Sections 3 and 4 as Hadoop applications on the Google&IBM cluster presented in Section 2. All the data sets used in this section are real spatial data sets supplied by the *High Performance Database Research Center* at Florida International University [12]. At the time of experimentation, there were jobs running in the cluster from other researchers that share this resource, thus some fluctuation in the results is expected.

### 5.1 R-Tree Construction

*Data sets and Setup*
Experiments are executed on two real spatial data sets. Data set descriptions are shown in Table 4. The points in the data sets are angular coordinates in (*latitude*, *longitude*) format. In the following experiments, we use the *Z-order* space-filling curve [9] as *U* function to map the two-dimensional points into a single dimension. We used 3% of each data set as sampling size *L* (see first phase of the algorithm in Section 3). Data sets are in tabular structured format (CSV), where each line represents an object. We used Hadoop supplied functions to read objects (text lines) from the data sets. During the second phase, Reducers build their individual R-Trees in-memory (to avoid high disk latencies in maintaining the tree along object insertions), then the trees are peristed on Hadoop distributed file system.

*Time Performance*
This experiment consists of building R-Tree indices on the Google&IBM cluster changing the parameter *R* in phase-2, that is, the number of concurrently-built R-Trees, from 2 up to 64. As *R* varies, job completion times are measured for Mappers and Reducers as well as quality statistics on the resulting R-Trees. As a reference, we also ran a single-process R-Tree construction on a dedicated local machine with Intel Xeon E7340 2.4GHz processor and 8GB of RAM running Windows OS; we could not run the single process in the cluster since we do not have login access to

**Table 4.** Spatial data sets used in experiments

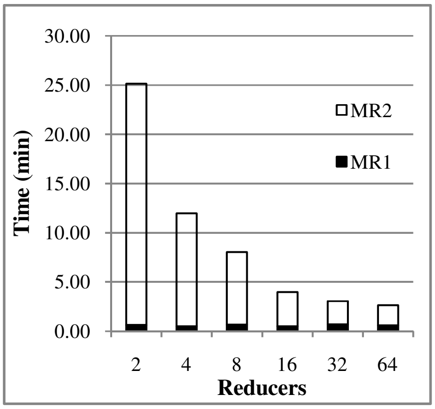
| Data set | Objects (millions) | Data size (GB) | Description |
|---|---|---|---|
| ***FLD*** | *11.4* | *5* | Points of properties in the state of Florida. |
| ***YPD*** | *37* | *5.3* | Yellow pages directory of points of businesses mostly in the United States but also in other countries. |

**Table 5.** MapReduce job completion times (in minutes) for the Phase 1 (*MR1*), and various Reducers (R) in Phase 2 (*MR2*) of building an R-Tree. Also, completion times for single-process (SP) constructions ran on a local machine are shown.

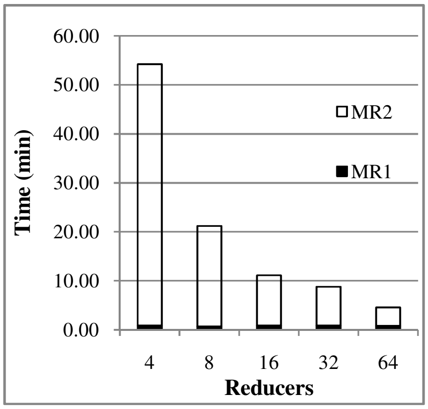
| | | **MR1:** ***Partitioning Function*** | | **MR2:** ***R-Tree Construction*** | | |
|---|---|---|---|---|---|---|
| **Data set** | **R** | **Map** | **Reduce** | **Map** | **Reduce** | **Total MR1+MR2** |
| ***FLD*** | 2 | 0.35 | 0.28 | 0.40 | 24.12 | 25.15 |
| | 4 | 0.28 | 0.23 | 0.40 | 11.07 | 11.98 |
| | 8 | 0.47 | 0.22 | 1.73 | 5.62 | 8.03 |
| | 16 | 0.30 | 0.22 | 0.40 | 3.05 | 3.97 |
| | 32 | 0.48 | 0.23 | 0.40 | 1.95 | 3.07 |
| | 64 | 0.28 | 0.33 | 0.45 | 1.60 | 2.67 |
| | SP | - | - | - | - | 27.34 |
| ***YPD*** | 4 | 0.47 | 0.38 | 0.47 | 52.57 | 53.88 |
| | 8 | 0.22 | 0.45 | 0.72 | 25.42 | 26.80 |
| | 16 | 0.40 | 0.43 | 0.38 | 8.93 | 10.15 |
| | 32 | 0.40 | 0.43 | 0.42 | 4.65 | 5.90 |
| | 64 | 0.40 | 0.42 | 0.88 | 2.55 | 4.25 |
| | SP | - | - | - | - | 63.98 |

individual nodes. Thus, cluster and single process times are not comparable due to dissimilar hardware.

Table 5 shows MapReduce job completion times for R-Tree construction phases 1 and 2 on both spatial data sets as well as for a single-process build (SP); for *YPD* we start at *R*=4 due to memory limitations in cluster nodes for building in-memory trees with less number of Reducers. We do not include phase-3 processing times since it is of little significance compared to the other phases. Phase-1 (partitioning function computation) takes very little time, which is expected since sorting *L*=3% of objects from a data set can be quickly done in memory by the single reducer in this phase; for our largest data set *YPD*, about 1 million elements are sampled. Our *Z-order* values are 8-byte sized elements, so around 8MB of RAM is needed to execute the sort, which is much less than the memory of each cluster node. Likewise, Mappers in phase-2 read data sequentially and execute inexpensive Z-order value computations on their inputs.



(a) *FLD* data set



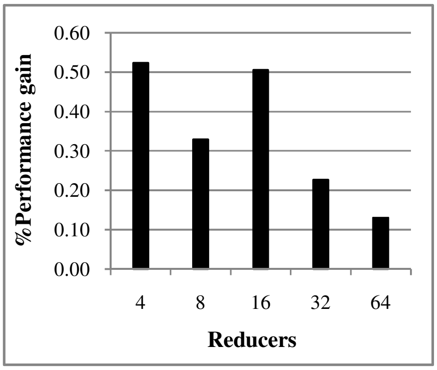
(b) *YPD* data set

**Fig. 4.** MapReduce job completion times for various number of reducers in phase-2 (MR2)

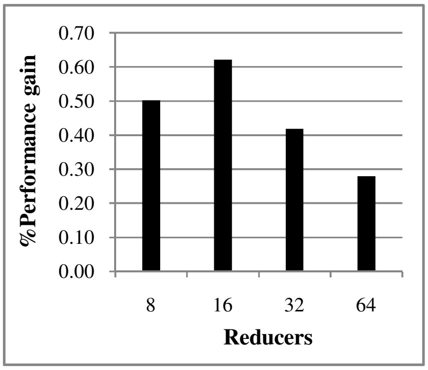
The most computationally intensive part is performed by Reducers in phase-2, where the actual R-Tree constrution occurs. The fewer the number of Reducers, the longer the R-Tree construction takes, since each task receives larger number of objects. Figure 4 shows job completion times as stacked bars of the map and reduce execution times. In this figure, almost linear scalability is observed as more parallelism is induced by increasing *R* in phase-2. As expected, the improvement rate is high for few Reducers but drops as the number of Reducers increases since partitioning overheads in phase-1 (*MR1*) start becoming significant compared to R-Tree build time in phase-2 (*MR2*). In fact, for larger values of *R*, the dominating time component is given by *MR1* which, as can be seen in Table 5, is almost constant for a given data set. Thus, much less improvements are expected as *R* is increased beyond 64.

Although we cannot compare our MapReduce and single process (SP) times due to mismatch in hardware, the MapReduce parallelization certainly yields performance benefits for large-scale data sets. For example, it takes more than an hour to sequentially build the *YPD* R-Tree, while in parallel the task can be achieved in less than 5 minutes with 64 Reducers. However, the resulting R-Trees are different due to differences in object insertion sequences. Later in this section we measure and discuss R-Tree quality parameters for both cases.

Figure 5 presents percentages of performance gains in job completion times in relation to subsequent increases in the number of Reducers in the second phase of the algorithm. For example, in the *YPD* dataset, going from 4 to 8 Reducers we observe 50% decrease in job completion time, which represents linear scalability. On the other hand, going from 8 to 16 Reducers shows super-linear gains (62%). We pressume this may be due to heterogeneous nodes in the cluster (eventually the job with R=16 was executed on faster nodes), or it may be the cluster resources were idler during that period. As discussed, as we increase the number of Reducers, performance gains are less significant because the execution time for the first phase, which has a sequential component (*Reduce*), stays almost constant.



(a) *FLD* data set



(b) *YPD* data set

**Fig. 5.** MapReduce job percentage of performance gains as the number of reducers is increased

*Quality of Generated R-Trees*

We use equations (2) and (3) below to compute the area and overlap metrics respectively for a given consolidated R-Tree with root *T*:

$$Area(T) = \sum_{i=1}^{n} Area(T_i.MBR) \tag{2}$$

$$Overlap(T) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} Area(T_i.MBR \bigcap T_j.MBR) \tag{3}$$

where $n$ is the number of children (small R-Trees generated by Reducers) of $T$, and $T_i$ is the $i$-th child node of $T$. Note that other metrics of R-Tree quality could be considered as well, e.g., consider all the nodes of the R-Tree instead of just the top level. Minimal area and overlap are known to improve search performance [13] since they increase path pruning abilities of R-Tree navigation algorithms.

Table 6 shows quality metrics on the consolidated R-Trees built for various number of Reducers and single process (SP); for reference, the U.S. Census Bureau reports Florida state land area roughly as 54,000 square miles as of 2000 [14]. As expected, we see the total MBR area and the overlap increase as the parallelism (*R*) increases because the construction of each small R-Tree is unaware of the rest of the data set, lowering the chance of co-locating neighbor objects within the same R-tree. This means that we degrade the R-Tree quality without gaining in execution time. The latter can adversely effect performance of search algorithms, such as nearest neighbor type of queries, due to extra I/Os incurred in traversing multiple sub-trees.

**Table 6.** Statistics on consolidated R-Trees built by various number of *Reducers* (*R*), and single process (*SP*) construction

| | | **Objects per Reducer** | | ***Consolidated* R-Tree** | | | |
|---|---|---|---|---|---|---|---|
| **Data set** | ***R*** | **Average** | **Stdev** | **Nodes** | **Height** | **Area (sq.mi)** | **Overlap (sq.mi)** |
| ***FLD*** | 2 | 5,690,419 | 12,183 | 172,776 | 4 | 132,333.9 | 304.4 |
| | 4 | 2,845,210 | 6,347 | 172,624 | 4 | 106,230.4 | 4,307.9 |
| | 8 | 1,422,605 | 2,235 | 173,141 | 4 | 103,885.8 | 17,261.9 |
| | 16 | 711,379 | 2,533 | 162,518 | 4 | 96,443.1 | 21,586.3 |
| | 32 | 355,651 | 2,379 | 173,273 | 3 | 140,028.7 | 80,389.1 |
| | 64 | 177,826 | 1,816 | 173,445 | 3 | 152,664.2 | 96,857.7 |
| | SP | 11,382,185 | 0 | 172,681 | 4 | 746,145.0 | 1,344,836.8 |
| ***YPD*** | 4 | 9,257,188 | 22,137 | 568,854 | 4 | 26,510,946.3 | 21,574,857.8 |
| | 8 | 4,628,594 | 9,413 | 568,716 | 4 | 23,160,080.0 | 20,480,729.6 |
| | 16 | 2,314,297 | 7,634 | 568,232 | 4 | 67,260,270.0 | 54,582,299.8 |
| | 32 | 1,157,149 | 6,043 | 567,550 | 4 | 68,626,854.9 | 54,008,538.5 |
| | 64 | 578,574 | 2,982 | 566,199 | 4 | 69,791,363.8 | 55,064,139.4 |
| | SP | 37,034,126 | 0 | 587,353 | 5 | 164,966,688.5 | 658,583,322.6 |

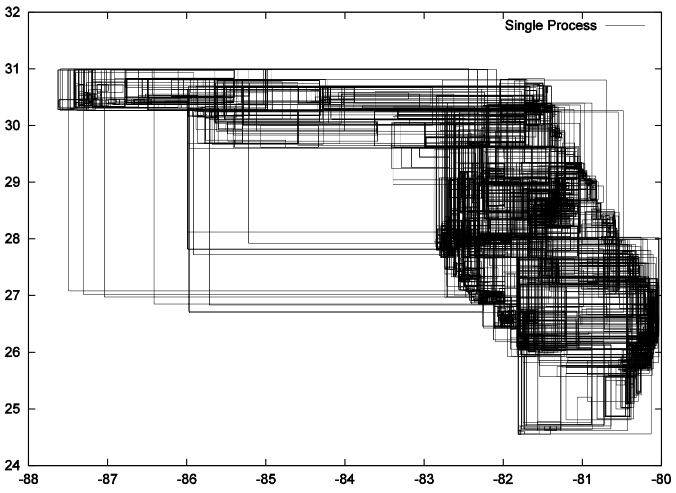


**Fig. 6.** MBR plotting for *FLD* data set on an R-Tree built by a single process

For a sequential construction (*SP*), we observe these metrics are much worse, especially the overlap factor, since objects are not spatially shuffled but rather inserted in the data set original sequence. Thus, higher performance penalties are expected in *SP* constructed R-Trees. On the other hand, the tree height slightly decreases for *FLD* for *R* beyond 32 because more small trees means that each one of them may be shorter, while for *YPD* the height increases by one level for the *SP* case. In general, small variations in tree height is less significant from a performance standpoint.

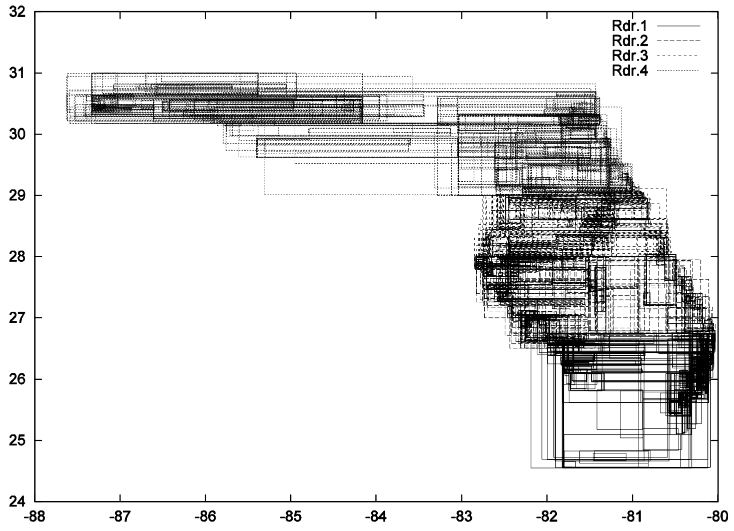


**Fig. 7.** MBR plotting for *FLD* data set for R-Tree built by MapReduce with *R*=4

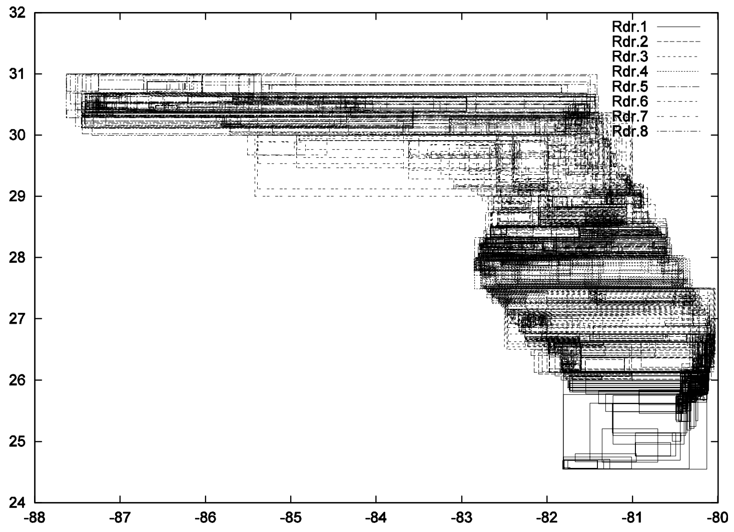


**Fig. 8.** MBR plotting for *FLD* data set for R-Tree built by MapReduce with *R*=8

To visually study the effect of increasing *R* over the MBR distribution, we have plotted the MBRs of the resulting R-Trees for the case of 4 and 8 Reducers in Figures 7 and 8 respectively for the Florida state data set (*FLD*). Also the same type of graph is shown in Figure 6 for the *SP* R-Tree. In neither case is the root MBR plotted since it is common for all trees.

A few observations can be made from the MBR plottings. First, the partitioning mechanism employed in our algorithms seems to be effective in preserving spatial locality. This results in individual Reducers indexing highly localized objects; their boundaries, however, result in multiple overlappings, which are inevitable. Second, as the number of *Reducers* is increased from 4 to 8, the plotting shape resembles more the actual shape of the Florida state; that is, *R*=8 reduces wasted areas (where no actual objects are located) as the Area statistic confirms in Table 6. In fact, Table 6 shows steady decrease in area from 2 to 16 *Reducers*; after that the area keeps on increasing. Third, when the R-Tree is built on the original sequence of objects (no object shuffling) in *SP* mode, large wasted areas are generated as can be observed in Figure 6. From a performance optimization perspective, MapReduce generated R-Trees seem to be better tuned than their single-process counterpart. Therefore, we see promising performance improvements in MapReduce generated R-Trees, which deserve closer verification.

## 5.2 Tile Quality

*Data set and Setup*

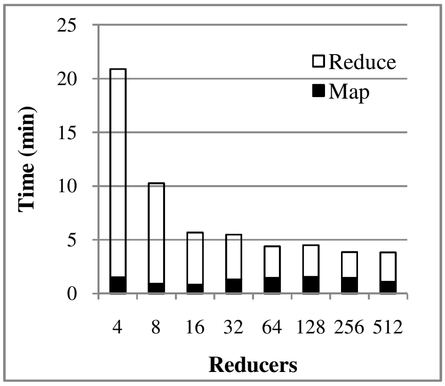
The data set used in the experiments is a 3-inch resolution aerial imagery of Miami Dade County of Florida. The size of the data set is about 52GB after compression. Imagery data is stored as compressed DOQQ file format. There are 482 compressed DOQQ files, each of which contains 4096 tiles. Each tile is 400 by 400 pixels and has 3 bytes for each pixel as the Red, Green and Blue channel. The size for each tile is 480,000 bytes uncompressed and compressed tile is about 50 KB each.

*Experiments*

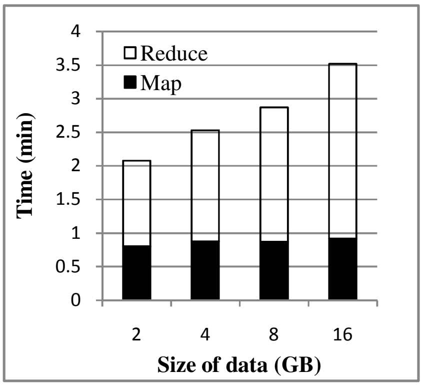
Two experiments are carried out for this data set. The first experiment uses a subset of the data set that is a re-sampled version of the original one. It is about 20GB and has 482 files with 1024 tiles each. The size of the files ranges from several megabytes to around 80 megabytes, and the number of Reducers is varied from 4 to 512. The second experiment uses different sized subsets of the original data set. The size of the files ranges from 2GB to 16GB, and the number of Reducers is fixed at 256.

In the first experiment, the number of Mappers is also fixed, determined by the data set size. Thus, the execution time of the map phase is similar through different runs, as can be seen in Figure 9 (a). The execution time slightly fluctuates because there were other concurrent jobs running in the cluster at the same time. As the number of Reducers increases, the execution time of the reduce phase largely decreases for smaller number of reducers, and less improvements are obtained for larger number of reducers. This is because the same amount of work is now shared by more Reducers. When the number of Reducers is larger than 64, the execution time of the reduce phase stabilizes to around 2.5 minutes. This could be explained by the launching time of Reducers dominating the whole time at this point. With 64 Reducers, each of them will be writing around 482/64 ≈ 8 files. The time taken to write 8, 4 (128 Reducers) or even less files is negligible compared with the launching time of that many Reducers.

In the second experiment, Figure 9 (b), as the size of the data set increases with constant number of reducers (256), the execution time of the map phase hardly changes, which is consistent with the data parallelization provided by the MapReduce



(a) Fixed data size, variable Reducers



(b) Variable data size, fixed Reducers

**Fig. 9.** MapReduce job completion time for tile quality computation

model, that is, more Mappers are engaged in processing the data. The execution time of the reduce phase increases because there are now more files to be written with the same number of Reducers.

## 6 Related Work

*Space-filling Curves*

The idea of using space-filling curves to map multi-dimensional spaces into a single dimension has been studied for the case of spatial databases [15, 8]; popular space-filling curves, such as Peano and Hilbert, have been studied in great level of detail. We used the Z-order curve in our experiments. This curve showed high spatial locality preservation for our experimented real data sets. Other curves can certainly be evaluated, which goes beyond our focus on the parallelization of two concrete spatial problems with MapReduce.

*Parallel R-Tree Constructions*

Previous works on R-Tree parallel construction have faced several intrinsic distributed computing problems such as data load balancing, process scheduling, fault tolerance, etc., for which they elaborated special-purpose algorithms. Schnitzer and Leutenegger [16] propose a Master-Client R-Tree, where the data set is first partitioned using Hilbert packing sort algorithm, then the partitions are declustered into a number of processors (via an specialized declustering algorithm), where individual trees are built. At the end, a master process combines the individual trees into the final R-Tree. Another work by Papadopoulos and Manolopoulos [17] proposed a methodology for sampling-based space partitionining, load balancing, and partition assignment into a set of processors in parallely building R-Trees. They also discuss some alternatives when the global (consolidated) index has imperfections such as different heights across individual R-Trees.

In MapReduce, these parallel computing concerns are abstracted out from the application logic, and managed transparently as part of the MapReduce framework. Further, all nodes in the cluster access a common distributed file system, with automatic fault-tolerance and load balancing support, where data locality is employed as base criterion to assign Mappers and Reducers (preferably) to nodes already containing the input data. In contrast, traditional parallel processing works assume every node has its own storage, in a shared-nothing type of architecture, where data transfer among nodes becomes an important optimization goal.

*MapReduce on Spatial Data*

MapReduce framework was used to solve another spatial data problem by Google [18], where they study the problem of road alignment by combining satellite and vector data. This work concentrates on the complexities of the problem, which are more challenging than the MapReduce algorithms.

Schlosser et al. [19] worked on building octrees in Hadoop for later use in earthquake simulations at large-scale. Their approach builds a tree in a bottom up fashion. The map function in the first iteration generates leaf nodes, then the reduce function coalesces homogeneous leaf nodes into a subtree. Subsequent iterations have identity functions in mappers, and successively use reduce functions to construct the final tree.

*Relationship to MPI*

Message Passing Interface (MPI) [20] is a specification of a language-independent communication model targeted at writing parallel programs, and it is widely used in a variety of computer cluster platforms. MPI libraries provide primitives and functionality for communication control among a set of processes. Typically, developers need to add explicit calls to synchronize processes and move data around. The key differences between MPI and MapReduce is that MapReduce exploits its simplified model to automatically parallelize tasks (Mappers and Reducers), hiding from programmers the need to worry about process communication, fault-tolerance, and scalability, which are transparently managed by key components, such as cluster management system and distributed file system, that the MapReduce framework is built-upon [6]. For example, for the R-Tree case study, the Java implementation of the Map and Reduce functions of the first phase, and Map of the second phase have each less than 40 lines of code. The Reduce function in the second phase has about 70 lines of code since it includes extra code for persisting the tree on the distributed file system and collecting build statistics. These numbers do not include application-specific routines, which are needed regardless of the parallel model.

In MapReduce, the underlying assumption is that the solution can be expressed in terms of the Map and Reduce functions working on key/value pairs. In some cases this may not be natural, such as relational joins or multi-stage processes, and can lead to inefficiencies. Then, MPI-like parallel implementations have more opportunities to address application-specific optimizations, due to its finer process control. However, high-level languages have been proposed to address this problem in MapReduce architectures by providing efficient primitives for massive data analysis combining SQL-like declarative style and MapReduce procedural programming [21][22].

## 7 Conclusions

In this paper, we used the MapReduce programming model to solve two important spatial problems on a Google&IBM cluster: (a) bulk-construction of R-Trees and (b) aerial image quality computation, which involve vector and raster data, respectively. The experimental results we obtained indicate that the appropriate application of MapReduce could dramatically improve task completion times. Our experiments show close to linear scalability. However, performance is not the only concern for R-Tree construction, which is sensitive to the ordering of objects in its input, but also the quality of the result. MapReduce generated R-Trees have improved quality in terms of MBR area and overlap measurements compared to the single-process construction counterpart. No such quality problem arises in the aerial image quality computation. Our experience in this work shows MapReduce has the potential to be applicable to more complex spatial problems.

## References

[1] Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. In: SIGMOD 1984, pp. 47–57 (1984)

[2] NSF Cluster Exploratory Program, http://www.nsf.gov/pubs/2008/nsf08560/nsf08560.htm

[3] Google&IBM Academic Cluster Computing Initiative, http://www.google.com/intl/en/press/pressrel/20071008_ibm_univ.html

[4] Apache Hadoop project, http://hadoop.apache.org

[5] Ghemawat, S., Gobioff, H., Leung, S.-T.: The Google file system. SIGOPS Operating Systems Review 37(5), 29–43 (2003)

[6] Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. In: Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation, vol. 6, p. 10. USENIX Association (December 2004)

[7] Asanoa, T., Ranjanb, D., Roosc, T., Welzld, E., Widmayer, P.: Space-filling curves and their use in the design of geometric data structures. Theoretical Computer Science 181(1), 3–15 (1997)

[8] Lawder, J.K., King, P.J.H.: Using Space-Filling Curves for Multi-dimensional Indexing. In: Jeffery, K., Lings, B. (eds.) BNCOD 2000. LNCS, vol. 1832, pp. 20–35. Springer, Heidelberg (2000)

[9] Morton, G.M.: A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing, Technical Report, Ottawa. IBM Ltd, Canada (1966)

[10] O'Malley, O.: TeraByte Sort on Apache Hadoop, Yahoo! (May 2008)

[11] Doqq file format, http://egsc.usgs.gov/isb/pubs/factsheets/fs05701.html

[12] High Performance Database Research Center (HPDRC), Research Division of the Florida International University, School of Computing and Information Sciences, University Park, Telephone (305) 348-1706, FIU ECS-243, Miami, FL 33199

[13] Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles 19(2), 322–331 (1990)

[14] U.S. Census Bureau, Florida State and County QuickFacts, `http://quickfacts.census.gov/qfd/states/12000.html` (last revised: July 25, 2008)

[15] Abel, D.J., Mark, D.M.: A comparative analysis of some two-dimensional orderings. International Journal of Geographical Information Science 4(1), 21–31 (1990)

[16] Schnitzer, B., Leutenegger, S.T.: Master-client R-trees: a new parallel R-tree architecture. In: Proceedings of the 11th International Conference on Scientific and Statistical Database Management, pp. 68–77 (August 1999)

[17] Papadopoulos, A., Manolopoulos, Y.: Parallel bulk-loading of spatial data. Parallel Computing 29(10), 1419–1444 (2003)

[18] Wu, X., Carceroni, R., Fang, H., Zelinka, S., Kirmse, A.: Automatic alignment of large-scale aerial rasters to road-maps, Geographic Information Systems. In: Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems, Article No. 17 (2007)

[19] Schlosser, S.W., Ryan, M.P., Taborda, R., Lopez, J., O'Hallaron, D.R., Bielak, J.: Materialized community ground models for large-scale earthquake simulation. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, Conference on High Performance Networking and Computing, pp. 1–12 (2008)

[20] Gropp, W., Lusk, E., Skjellum, A.: Using MPI: Portable Parallel Programming with the Message-Passing Interface. MIT Press, Cambridge (1999)

[21] Yang, H.-c., Dasdan, A., Hsiao, R.-L., Parker, D.S.: Map-reduce-merge: simplified relational data processing on large clusters. In: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pp. 1029–1040 (2007)

[22] Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig latin: a not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 1099–1110 (2008)

# Optimization and Execution of Complex Scientific Queries over Uncorrelated Experimental Data

Ruslan Fomkin and Tore Risch

Department of Information Technology, Uppsala University,
Box 337, SE-75105 Uppsala, Sweden
{Ruslan.Fomkin,Tore.Risch}@it.uu.se

**Abstract.** Scientific experiments produce large volumes of data represented as complex objects that describe independent events such as particle collisions. Scientific analyses can be expressed as queries selecting objects that satisfy complex local conditions over properties of each object. The conditions include joins, aggregate functions, and numerical computations. Traditional query processing where data is loaded into a database does not perform well, since it takes time and space to load and index data. Therefore, we developed SQISLE to efficiently process in one pass large queries selecting complex objects from sources. Our contributions include runtime query optimization strategies, which during query execution collect runtime query statistics, reoptimize the query using collected statistics, and dynamically switch optimization strategies. Furthermore, performance is improved by query rewrites, temporary view materializations, and compile time evaluation of query fragments. We demonstrate that queries in SQISLE perform close to hard-coded C++ implementations of the same analyses.

## 1 Introduction

Large volumes of data produced and shared within scientific communities are analyzed by many researchers to test scientific hypotheses. For example, in High Energy Physics (HEP) a lot of data is generated by simulation software from the Large Hadron Collider (LHC) experiment ATLAS [4]. The data is stored in files and describes effects from collisions of particles.

This paper investigates the use of query processing techniques to implement such scientific applications. Data is represented as complex objects describing measurements of independent real-world events. The analyses are selections of events applying conjunctions of complex numerical filters on each complex object.

In ATLAS, a *collision event* generates measurements of new particles summarized as a complex object. Generated objects are stored in files, which are read by the data management library ROOT [7]. Since every collision is performed independently from others, each complex object describing an event is analyzed separately and there are no joins between data from different events. Therefore each ROOT file can be processed in one pass per query as a stream of objects describing collision events. Physicists test their hypotheses on these data by selecting interesting events. An example of a scientifically interesting event is a collision event which is likely to

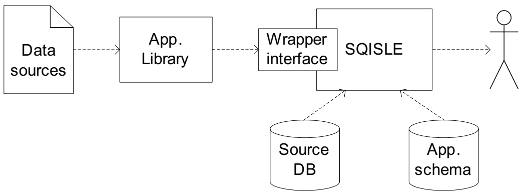


**Fig. 1.** Architecture of SQISLE with data flow

produce Higgs bosons [5,11]. A collision event is interesting if it satisfies some conditions, called *cuts*, specified over the object describing the event. The cuts are complex conditions over properties of each object involving joins, aggregate functions, and numerical computations.

Currently physicists test their hypotheses using regular programming languages, e.g., C++. The analysis programs retrieve descriptions of events from files through specific data management libraries, e.g. ROOT. However, it takes lots of efforts for physicists to code their analyses as C++ programs. Furthermore, good knowledge of programming methodologies is necessary for writing extensible and understandable programs for complex analyses.

We present a database approach to test scientific hypotheses as declarative conjunctive queries. We found that while such queries can be handled using traditional query processing techniques, performance is very poor due to slow data load and index times, space overhead of indexed data, and poor cost estimates for large queries with many aggregates and user-defined numerical functions. On the other hand, such queries can be processed very quickly using hand-coded C++ programs, but each program typically takes a scientist weeks to create. To improve performance while retaining ease of query specification, we created SQISLE (*Scientific Queries over Independent Streamed Large Events*), a query processing system that takes advantage of the special data and query characteristics of our target domain (high-energy physics) while also meeting its unique challenges. SQISLE employs a one-pass *streaming approach* to process queries where data stays in their sources, e.g. ROOT files, and is streamed through the system. SQISLE provides facilities for complex queries over streams of objects with complex structure, as required for our kind of scientific applications. The system reads complex objects from sources, e.g. description of collision events from ROOT files, through a wrapper interface and processes the objects one-by-one as they are streamed. The objects are thus analyzed in one pass by reading data sequentially without populating a database. This streaming approach requires limited memory and is shown to be efficient compared to the traditional *loading approach* where data is loaded into a database and indexed before being queried.

Instead of relying on static query optimization, SQISLE collects query statistics at runtime, uses them to reoptimize the query, and dynamically switches optimization strategies. SQISLE's performance is further improved by query rewrites, temporary view materializations, and compile time evaluation of query fragments. We show that queries in SQISLE perform close to or better than equivalent C++ implementations hand-coded by scientists.

SQISLE extends Amos II [24]. The architecture of SQISLE is illustrated by Fig. 1, where the arrows show the data flow during query execution. A scientist specifies an analysis as a query over a stream of complex objects from *data sources* processed by SQISLE through a *wrapper interface*. The scientists write their analysis queries in terms of a high level application schema (*App. schema*) that defines the structure of the streamed complex objects. The source database (*Source DB*) contains meta-data about stream sources. It is accessed in queries to locate sources containing data for the analyses and their meta-data. The wrapper interface is defined in terms of an application data management library (*App. Library*), e.g., ROOT.

To obtain efficient execution plans for complex queries over streams of complex objects, SQISLE uses runtime query optimization strategies implemented by a *profile-controller* operator. It encapsulates in each query the fragment that tests complex conditions (hypotheses) over properties of the streamed complex objects. During runtime it controls collecting statistics for the fragment, reoptimizes the fragment based on collected statistics, and dynamically switches optimization strategies. The cost-based query optimization utilizes a cost model for aggregate functions over nested subqueries from [9]. To alleviate estimation errors in large queries the fragments are decomposed into conjunctions of subqueries over which runtime statistics are measured [9].

Data from controlled scientific experiments produced with the same experimental run conditions usually have similar statistical properties. Therefore, we assume that stream data statistics are similar. For example, in the ATLAS application properties of events have the same distribution for events generated in the same experiment. Once a fragment is reoptimized based on sufficient sampled data from the beginning of the queried stream, the query execution is immediately continued using the reoptimized query execution plan for the rest of the stream without profiling overhead.

The query processing is further improved by *query rewrites*, use of *materialized views*, and *compile time evaluation*. Query rewrite rules reduce the number of predicates in queries. Views are materialized once per read complex object and the materialized results are accessed while processing the same object and then discarded. Compile time evaluation [15,23] executes some predicates of a query at compile time before query execution and replaces predicates with execution results.

By evaluating the performance of SQISLE for ATLAS queries over ROOT files with different selectivities, we show that the SQISLE query processing techniques improve performance of queries substantially. The query performance is compared with the performance of a manually coded C++ program provided by the physicists doing the same analysis. The SQISLE implementation is shown to have performance close to or better than the hard-coded C++ implementation. Ideally a C++ program should perform better than declarative programs interpreted by a DBMS, but in practice writing an efficient C++ programs requires substantial effort, which has to be repeated for new queries or for data from new experiments. Using a declarative query language for testing scientific hypotheses is thus much more efficient for research productivity than expressing the tests as more complex C++ programs.

In summary, the contributions are:

- One-pass query processing algorithms are shown to provide efficient implementations of declarative queries testing our kind of scientific hypotheses.

- Runtime query optimization of streaming queries by the profile-controller operator allows measuring real query behavior and dynamically switching execution strategies. The runtime strategies are shown superior to a static cost-based approach.
- With the proposed runtime optimization strategies, the streaming approach is shown to be more efficient than the loading approach for the targeted kind of queries with performance close to a hard-coded C++ program.

The rest of the paper is organized as follows. Section 2 presents how an ATLAS analysis is specified as a query. The SQISLE query processing techniques are presented in Section 3. The query performance is evaluated in Section 4. Section 5 presents related work. The paper is concluded in Section 6.

## 2 The ATLAS Application Queries

To evaluate the query processing techniques implemented in SQISLE an ATLAS application is defined as SQISLE queries. For example, the hypothesis from [5], which searches for events producing Higgs bosons, is specified as this query:

```
1:    select e
2:    from Event e, EventFile f
3:    where name(experiment(f)) = "bkg2" and
4:          fileid(f) < 15 and
5:          e in events(filename(f)) and                 (1)
6:          hadrtopcut(e) and jetvetocut(e) and
7:          misseecuts(e) and zvetocut(e) and
8:          threeleptoncut(e) and leptoncuts(e);
```

The query selects objects of type *Event* satisfying six cuts constituting the hypothesis from [5] called the *Six Cuts Analysis*. On lines 3-5 the query specifies the sources to query by selecting the files produced by the experiment named *bkg2*. The source database is searched in lines 3-4, while the function *events* calls a *ROOT wrapper interface function* to read objects from the selected ROOT files. The rest of the query specifies the *Six Cuts Analysis*.

Wrapper interface functions read data from sources as complex *stream objects* represented by a data type named *Sobject*. The stream objects are defined as user-defined types, and are deallocated automatically and efficiently by an incremental garbage collector when they are not referenced any more.

Each cut is a view defined as a Boolean function that returns true if the cut is fulfilled. The views are defined as declarative queries over properties of each event object *e* involving joins, aggregate functions, and complex numerical computations defined in terms of a high-level application schema. The type hierarchy of the application schema for the ATLAS application is presented in Fig. 2.

A wrapper interface function reads events and instantiates them as stream objects of type *Event*, a subtype of type *Sobject*. Each complex event object describes measurements from one collision. Objects of type *Particle* represent various kinds of particles produced by the collision, which are derived from the event object by *transformation views*. The transformation views are defined as declarative functions mapping stream objects returned by a wrapper interface function into a set of derived stream objects representing objects in terms of the application schema.

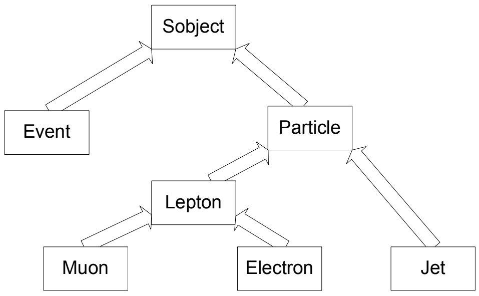


**Fig. 2.** Type hierarchy of an application schema for the ATLAS application

For example, the *Three Lepton Cut*, one of the simplest among the six cuts being part of the *Six Cuts Analysis*, requires that an event has exactly three *isolated leptons* with $|Eta|<2.4$ and $Pt>7$, where at least one lepton has $Pt>20$. The *Three Lepton Cut* is defined in SQISLE as this Boolean function:

```
create function threeLeptonCut (Event e) -> Boolean as
select TRUE
where  count(isolatedLeptons(e)) = 3
       and some( select r
                 from Real r
                 where r = Pt(isolatedLeptons(e))
                       and r > 20.0);
```

The function *isolatedLeptons* is defined as:

```
create function isolatedLeptons(Event e) ->
                                Bag of Lepton as
select l
from Lepton l
where l in leptons(e)
      and pt(l) > 7.0
      and abs(eta(l)) < 2.4;
```

The functions *Pt* and *Eta* are defined in terms of numerical operators calculating $Pt = \sqrt{x^2 + y^2}$ and $Eta = 0.5 \cdot \ln\left(\frac{\sqrt{x^2+y^2+z^2}+z}{\sqrt{x^2+y^2+z^2}-z}\right)$, respectively, over momentum $(x,y,z)$ of a particle. The function *leptons* is a transformation view defined as a query that returns all leptons detected by event *e*. It is defined as the union of the transformation views *electrons* and *muons*, which generate objects of types *Electron* and *Muon*, respectively. There is a detailed description of the used cuts and view definitions in [8].

Given that the cuts are defined as Boolean functions, a user query always contains the following kinds of query fragments:

- A *source access query fragment*, e.g. lines 3-5 in query (1), specifies sources to access and calls a *stream function* that emits a stream of complex objects.
- A *processing query fragment*, e.g. lines 6-8 in query (1), specifies the scientific analyses (tested hypotheses) as queries over views of complex objects, e.g. specifying cuts over events. The views are defined in terms of transformation views.

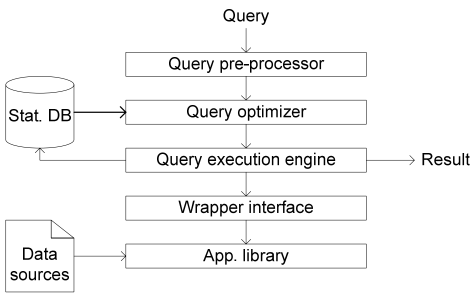


**Fig. 3.** Query processing steps in SQISLE

## 3 Query Processing in SQISLE

In [9] we implemented the above queries using the traditional loading approach where the data streams were first loaded into the DBMS and indexed before the queries were given. The loading and indexing is very time consuming. In our example, it takes about 15 seconds to load one ROOT file containing 25000 events, while the analysis alone of the 25000 events takes just 1.5 seconds, i.e. a total processing time of 16.5 seconds. Furthermore, the loading approach requires sufficient memory to store all complex objects indexed.

To avoid the high loading and indexing costs, in SQISLE, instead of preloading the data into a DBMS, the data stays in their sources, e.g. ROOT files, and are processed in one pass. The system reads the complex objects from the sources through a wrapper interface where each independent complex object is analyzed one-by-one as they are streamed.

Fig. 3 illustrates the query processing steps in SQISLE. The *query pre-processor* expands views and applies rewrite rules on the query. The cost-based *query optimizer* produces an execution plan, which is interpreted by the *execution engine*. The execution plan contains operators that call a *wrapper interface* implemented in terms of an application data management library (*App. Library,* e.g. ROOT) to access the *data sources*. To improve query processing, runtime query optimization collects data statistics for the query optimizer, which is stored in the *statistics database* (*Stat. DB*). The execution engine calls the optimizer to reoptimize the query at runtime using the collected statistics.

A general structure of an execution plan for a typical query like query (1) is presented in Fig. 4. With runtime query optimization the query pre-processor first splits the query into a source access query fragment and a processing query fragment. The figure illustrates the two corresponding subplans: the *source access plan* and the *processing plan*. The source access plan first calls *wrapper argument operators* that access the source database (*Source DB*) and the application schema meta-data (*App. schema*) to select sources and bind parameters $a_1$, $a_2$, … for a *wrapper interface operator*. The wrapper interface operator accesses each selected *data source*, e.g. a ROOT file, and generates a stream of complex objects, each bound to a *stream object variable, o*. The processing plan implements selections of *o* based on *analysis operators* and *aggregated subqueries* over properties of *o*. The aggregated subqueries apply aggregate functions on nested subqueries. The nested subqueries first call

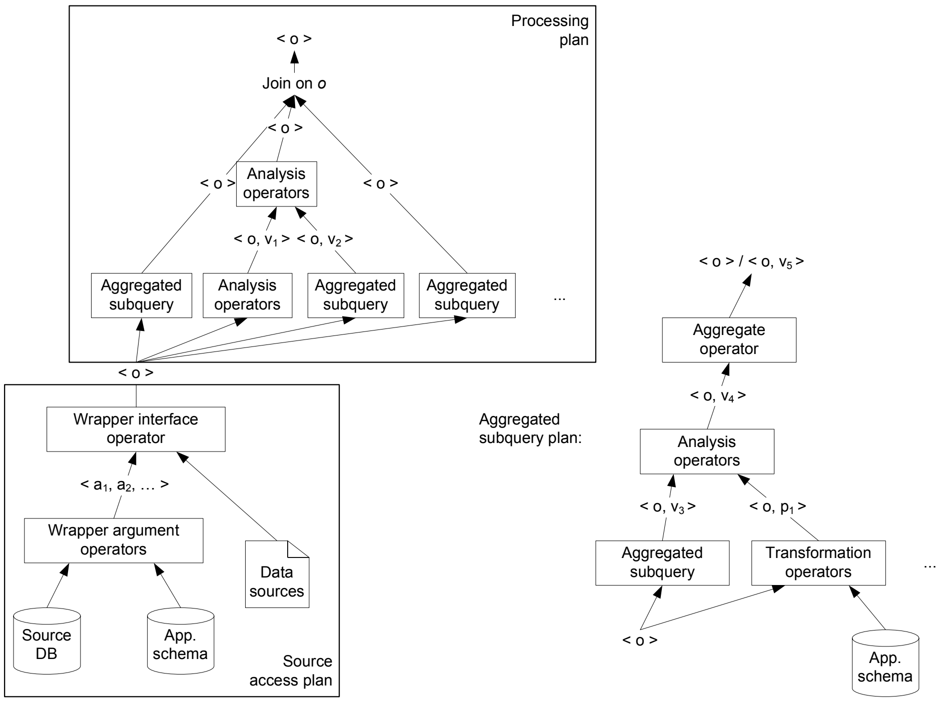


**Fig. 4.** General structure of a query plan

transformation views to derive stream objects in terms of the application schema and then analyze properties of the derived objects. The source access plan and the processing plan are joined only on stream object variable *o*.

For query (1) the number of operators in the source access plan is 10. It produces a stream of objects representing events, which are analyzed by the processing plan. The processing plan contains 22 operators and 8 of them are calls to aggregated subqueries. The number of operators in the aggregated subquery plans is between 9 and 59, including transformation operators and analysis operators implementing the selections. Some aggregated subqueries contain calls to further aggregated subqueries.

There are many possible operator orders for a processing plan. Thus query optimization is difficult, and the query plans obtained with naïve static cost-based query processing strategies are slow. Therefore SQISLE implements runtime query optimization to collect data statistics from streams at runtime in order to adapt the query plan using collected statistics. Runtime query optimization is managed by a special operator, the *profile-controller*. During query execution it monitors whether sufficient statistics have been collected so far while processing the stream. If so, it dynamically reoptimizes the query and switches to non-profiled execution by disabling collecting and monitoring statistics.

The runtime query optimization is investigated with three strategies:

1. *Attribute statistics profiling* maintains detailed statistics on the sizes of vectors stored in each stream object attribute as the objects are read. Once the sample size is large enough the query is reoptimized using the collected statistics.

2. *Group statistics profiling* first decomposes the queries into fragments, called *groups*, which are joined only on the stream object variable, and then maintains runtime statistics of executing each group. When sufficient statistics is collected the query is reoptimized.
3. *Two-phase statistics profiling* combines the two strategies above by in a first phase collecting detailed statistics of attribute vector sizes of stream objects to optimize the group definitions, and in a second phase switching to group statistics profiling for ordering the groups.

All three strategies assume that data statistics over the stream is *stable* so that the statistics collected in the beginning of the stream is expected to be close to the statistics for the entire data stream. This is the case in scientific applications such as the ATLAS experiment, since all collision events in a stream are generated with the same experimental run conditions. The strategies perform statistics sampling at runtime until the statistics are stabilized. Cost-based query optimization utilizes a cost model for aggregate functions over nested subqueries [9]. This *aggregate cost model* estimates costs and selectivities for aggregate functions from costs and selectivities of their nested subqueries.

## 3.1 The Profile-Controller Operator for Runtime Query Optimization

The query pre-processor modifies the view expanded query to add the profile-controller operator and encapsulate the processing query fragment with the operator. The processing query fragment needs to be optimized carefully, since it is defined as a large condition over a complex stream object. The optimization at runtime of the encapsulated complex condition is controlled by the profile-controller operator.

The profile-controller performs the following operations for each stream object *o*:

1. It executes the processing plan parameterized by *o*.
2. It checks if profiling is enabled. If so it calls a subroutine, the *switch condition monitor*, which supervises collection of data statistics. The switch condition monitor returns true if sufficient statistics is collected. To enable different kinds of profiling the switch condition monitor can be different for different strategies and can also be dynamically changed during query execution.
3. If item two is satisfied it calls another subroutine, the *switch procedure*, which reoptimizes the processing query fragment and either switches to another runtime query optimization strategy or disables profiling. The switch procedure is also dynamically replaceable.
4. The result of the processing query fragment in item one is always emitted as result of the profile-controller operator.

## 3.2 Attribute Statistics Profiling

When attribute statistics profiling is enabled, detailed statistics on stream object attribute vector sizes are collected when each new stream object *o* is emitted by the wrapper interface operator. The means and variances of the attribute vector sizes of *o* are maintained in an internal table.

The switch condition monitor maintains statistics to check for every tenth read stream object *o* whether an estimated mean attribute vector size $\overline{x}$ is close enough to the actual mean attribute vector size with probability *1-α*. The following *attribute switch condition* is checked:

$$z_{\alpha/2} \cdot S_E \leq \delta \cdot \bar{x} \tag{2}$$

The closeness is defined by *δ*. *α* and *δ* are provided as tuning parameters. The estimate of the mean $\bar{x}$ is calculated by $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ , where $x_i$ is the attribute vector size for the $i^{th}$ stream object, and *n* is the number of stream objects read so far. $S_E$ is an estimate of $\sigma / \sqrt{n}$ and calculated by $S_E = \sqrt{\frac{1}{n^2}\sum_{i=1}^{n} x_i^2 - \frac{\bar{x}^2}{n}}$ .

If the attribute switch condition is satisfied for every attribute, the switch procedure is called. It reoptimizes the processing query fragment and disables profiling. After this, when the wrapper interface operator constructs a new stream object, it does not collect statistics any more. The profile-controller executes only the processing plan and does not call the switch condition monitor or the switch procedure.

When the query is started there no statistics collected and the query is initially optimized using *default statistics* where the attribute vector sizes of stream objects, e.g. the number of particles per event, is approximated by a constant (set to nine).

## 3.3 Group Statistics Profiling

With *group statistics profiling*, first a *stream fragmenting algorithm* is applied to the query. The algorithm decomposes the processing query fragment into smaller query fragments called *groups* [9]. The groups have only the stream variable *o* in common and thus the groups are equi-joined only on *o*. The complex object *o* is selected by the query if it satisfies the inner join of all groups.

After optimization, each group is implemented by a separate *group subplan*, which is encapsulated by a *group monitor* operator. The group monitor operator takes a group subplan and a stream object as arguments and returns the result of applying the subplan on the stream object. If profiling is enabled, it measures execution time and selectivity of the monitored subplan.

The switch condition monitor calls the query optimizer at runtime for every read stream object *o* to greedily order the executions of the monitored subplans based on available statistics on the groups. An internal table keeps track of the groups and their statistics. The switch condition is true if the order of the groups in the new processing plan is the same for a number of read stream objects in a row, called the *stable reoptimization interval* (*SI*), which is provided as a tuning parameter.

The contents of the groups and the initial join order of the groups are optimized using the default statistics before starting to execute the query.

The profile-controller operator encapsulates the entire processing plan containing all the joined groups. It invokes the dynamically optimized query processing plan at runtime. If some join fails, the entire processing plan fails. Thus, to answer the query the processing plan must execute only those first group subplans up to the first subplan that fails. No group subplans joined after the failed one need to be executed to answer the query. However, statistics still should be collected for all groups, even those that need not be executed. Thus, if profiling is enabled, the switch condition monitor executes also those remaining groups that were not executed to answer the query. In this way statistics is first collected for all groups by the switch condition monitor. The groups are then greedily reordered based on the measured estimates of the group costs and selectivities. To minimize overhead the processing plan is reoptimized once for every read complex stream object rather than for every query plan operator; there is no dynamic reordering per operator as with Eddies [3].

### 3.4 Two-Phase Statistics Profiling

As with group statistics profiling, with *two-phase statistics profiling* queries are first fragmented into groups before executing them. To collect runtime statistics for optimizing of group subqueries, attribute statistics profiling is enabled initially when query execution is started. When the attribute switch condition (2) is satisfied, the entire query is reoptimized, including the groups, and attribute statistics profiling is disabled. Then the switch condition monitor and switch procedure are changed to perform group statistics profiling to produce a further optimized group join order.

The main advantage with the two-phase statistics profiling is that it enables optimization of group subqueries based on collected attribute statistics. With group statistics profiling alone, where the attribute values are not monitored, the groups themselves must be optimized based on heuristic default statistics.

### 3.5 Query Rewrite Strategies

The performance is measured comparing runtime query optimization with a manually coded C++ program. It will be shown that optimized query plans of selective queries may perform better than a C++ implementation, while non-selective queries are still around 28 times slower.

In order to improve the performance of non-selective queries, their performance bottlenecks were analyzed. It was found that most of the time is spent on computing the transformation views many times for the same stream object. To remove this bottleneck, the use of rewrite rules to speed up the queries is investigated. One kind of rewrite is based on observing that the derivation of particle objects from event objects can be regarded as a two-dimensional matrix transposition. Different variants of operators for the transposition were implemented and evaluated. The chosen matrix transpose operator generates new particle stream objects as the result of the transposition and temporarily caches them as an attribute on the currently processed event object. This strategy is called *transformation view materialization*. It improves performance of non-selective queries about 1.5 – 2.5 times compared with only runtime query optimization.

Queries are further simplified in SQISLE by removing unnecessary vector constructions in queries and view definitions. Some vectors are first constructed out of variables and then only specific element values are accessed explicitly; the constructions of such vectors are removed and the original variables are instead accessed directly without vector construction and access overheads. These *vector rewrites* improve performance of non-selective queries with factor 1.5 – 2.

In addition *computational view materialization* improve query performance by temporarily saving on each processed complex object the results of numerical calculations computing properties of derived stream objects used in analysis queries, e.g. in cut definitions. This pays off when a query does the same complex numerical calculations several times per complex object. Materialization of computational views improves non-selective queries with at least another 32%.

Finally, the performance of queries is further improved by compile time evaluation [15,23], which is a general technique to evaluate predicates at query compilation time and replace them with computed values. Compile time evaluation is used to remove accesses to application schema meta-data, which simplifies the queries. Compile time evaluation improves performance of non-selective queries an additional 20%

All together the above query rewrite techniques improve performance of non-selective queries around 5 times. The execution is still about 4 times slower than C++. However, the execution plan is currently interpreted in SQISLE and further performance improvements can be made by making an execution plan compiler. This is expected to make the plan as fast as C++ also for non-selective queries.

## 4 Performance Evaluation

Performance experiments are made for scientific analyses expressed as queries in SQISLE for the ATLAS application. The experiments are run on a computer having 2.8 GHz Intel P4 CPU with 2GB RAM and Linux OS.

The performance is evaluated with different query processing strategies for two different kinds of queries implementing *Six Cuts Analysis* [5] and *Four Cuts Analysis* [11]. The performance of the C++ implementation is measured only for *Six Cuts Analysis*, since this implementation was the only one provided by the physicists.

Data from two different ATLAS experiments stored in ROOT files were used. The experiment *bkg2* simulates background events, which unlikely produce the Higgs bosons, so the analysis queries are very selective (*Six Cuts Analysis* has selectivity 0.018% and *Four Cuts Analysis* has selectivity 0.19%). The experiment *signal* simulates events that are likely to produce Higgs bosons, and both kinds of queries over these data are non-selective (*Six Cuts Analysis* selects 16% events and *Four Cuts Analysis* selects 58% events).

Event descriptions from the *bkg2* experiment are stored in 41 ROOT files, where each file contains 25000 event objects, i.e. a 1025000 event objects in total. Event descriptions from the *signal* experiment are stored in a single file with 8623 event objects. The sizes of the event streams are scaled by reading subsets of the files.

Two different kinds of queries are measured for the two different experiments. *Six Cuts Analysis* uses the views *bkgsixcuts* and *signalsixcuts* for experiment *bkg2* and experiment *signal*, respectively. *Four Cuts Analysis* uses the views *bkgfourcuts* and *signalfourcuts,* which are less complex than *bkgsixcuts* and *signalsixcuts*. A view parameter is used to specify the number of events to read and analyze, i.e. the stream size. The details can be found in [8].

Two kinds of measurements are made: the *total query processing time* and the *final plan execution time*. The total query processing time is the total time for optimization, profiling, and execution of a query. The final plan execution time is the time to execute the optimized plan.

## 4.1 Evaluated Strategies

The following strategies are evaluated:

- *Naïve query processing (NaiveQP).* As a reference point, this strategy demonstrates performance of naive query processing without cost-based optimization. The cuts are executed in the same order as they are specified in the queries.
- *Static query processing with the aggregate cost model (StatQP).* This reference strategy demonstrates the impact of regular static cost-based optimization based on the aggregate cost model. The aggregate cost model is enabled, but no run-time query optimization strategies. No data statistics is available when the query is optimized and default statistics are used. Since queries are very large, they are optimized using randomized optimization [14,27,22], which is able to find a good plan in terms of estimated costs. The strategy is compared with *NaiveQP* to demonstrate impact of the static cost-based query optimization using default statistics.
- *Attribute statistics profiling (AttrSP).* The query is initially optimized with the aggregate cost model and default statistics. During execution of the query the statistics on sizes of the attribute vectors is collected and query reoptimization is performed using collected statistics. The initial optimization uses a fast greedy optimization method [16,18] and default statistics. The query reoptimization uses slow randomized optimization, which produces much better plan in terms of estimated cost than the greedy optimization.
- *Group statistics profiling (GroupSP).* After query fragmentation into groups, the created groups and their order are initially re-optimized per event by greedy optimization using default statistics until a stable plan is obtained. Fast greedy optimization is also used to reoptimize the group order since dynamic programming [26] produced the same execution plans.
- *Two-phase statistics profiling (2PhaseSP).* Greedy optimization is used first to produce optimized group subplans after performing attribute statistics profiling. In a second phase *GroupSP* is applied to optimize the order of the group subplans.
- *Full query processing (FullQP).* This strategy implements query rewrites combined with the best of the above optimization strategies.

As reference *FullQP* is compared with the following C++ implementations of the same analysis:

- *Unoptimized C++ implementation (NaiveCPP).* This strategy demonstrates the performance of a manual C++ implementation of *Six Cuts Analysis* executed in the same order as in query *bkgsixcuts*. This strategy is compared with *FullQP*.
- *Optimized C++ implementation (OptCPP).* This strategy demonstrates the performance of *Six Cuts Analysis* implemented in a C++ program where the order of the cuts had been optimized by a researcher manually. This strategy is compared with *FullQP*.

All evaluated strategies are summarized in Table 1.

**Table 1.** Evaluated query processing techniques

| Strategy | Aggregate cost model | Attribute statistics profiling | Group statistics profiling | Rewrites |
|---|---|---|---|---|
| NaiveQP | – | – | – | – |
| StatQP | + | – | – | – |
| AttrSP | + | + | – | – |
| GroupSP | + | – | + | – |
| 2PhaseSP | + | + | + | – |
| FullQP | + | + | + | + |
| NaiveCPP | C++ implementation with suboptimal order of cuts | | | |
| OptCPP | C++ implementation with the cuts ordered manually by a scientist | | | |

## 4.2 Experimental Results

The performance of different optimization approaches without query rewrites is investigated first. Then the additional impact of the query rewrites is investigated. Finally, the best strategy is compared with hardcoded C++ implementations. The sizes of input streams in the evaluations are scaled over six points per experiment as shown in Figures 5 and 6.

Fig. 5(a) presents performance of the query plans measured by the different optimization approaches for the selective complex query *bkgsixcuts* (0.018% events selected).

The query plan of the unoptimized processing strategy (*NaiveQP*) performs substantially worse than the other strategies. Static query optimization with the aggregate cost model (*StatQP*) gives a query plan that performs four times better than the query plan from *NaiveQP*. This demonstrates the importance of the aggregate cost model to differentiate between different aggregated subqueries.

The query plan obtained with attribute statistics profiling (*AttrSP*) performs twice better than the statically optimized plan (*StatQP*). This shows that runtime query optimization is better than static optimization.

The query plans from the group statistics profiling (*GroupSP*) and two-phase statistics profiling strategies (*2PhaseSP*) perform best and substantially better than the strategies without grouping. They outperform naïve query processing (*NaiveQP*) with a factor 450 and attribute statistics profiling without grouping (*AttrSP*) with a factor 50. This demonstrates that the grouped strategies *GroupSP* and *2PhaseSP* alleviate the problem of errors in the cost estimates [12] by measuring real execution time and selectivity for each group. The difference between *GroupSP* and *2PhaseSP* is insignificant (Fig. 5(b)). The total query processing times for the strategies (Table 2) demonstrate that *2PhaseSP* performs better than *GroupSP*. Thus *2PhaseSP* is chosen to optimize rewritten queries in *FullQP*. Fig. 5(b) demonstrates that the strategy with rewrites (*FullQP*) performs 17% better than the optimization strategies without rewrites (*GroupSP* and *2PhaseSP*) for the selective query *bkgsixcuts*. The query performance for the other selective query *bkgfourcuts* is similar to query *bkgsixcuts*, but with lower overheads, since the query is simpler.

Fig. 5(c) demonstrates that for the selective query the best query processing strategy (*FullQP*) performs 20% better than unoptimized C++ (*NaiveCPP*). However, the C++ implementation where the order of cuts is optimized manually by the physicist, *OptCPP*, performs 34% better than the query plan from *FullQP*. Further performance

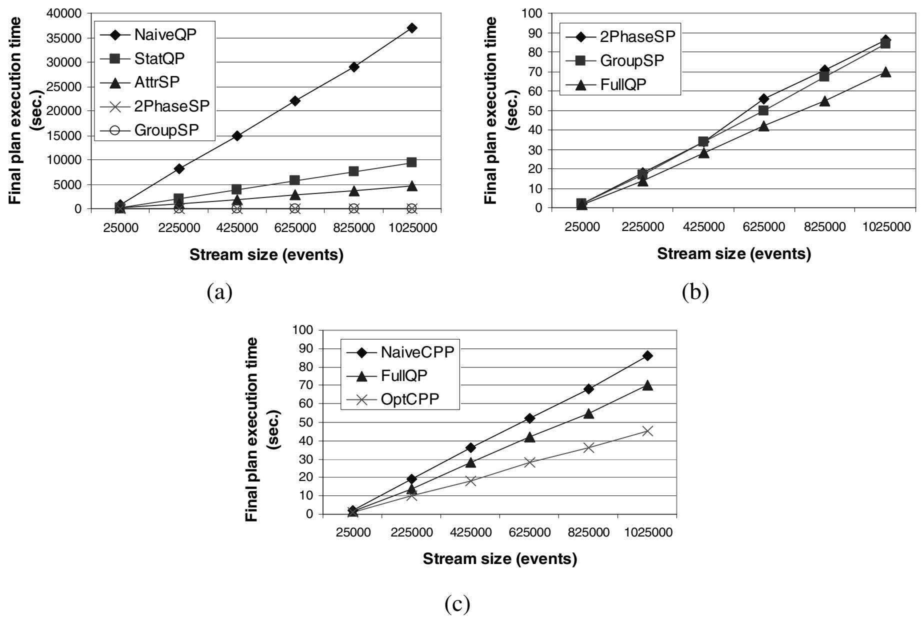


**Fig. 5.** Performance of different strategies for selective complex query *bkgsixcuts*: (a) and (b) show performance of different query strategies in SQISLE, while (c) compares performance of the best strategy (*FullQP*) with the C++ implementations

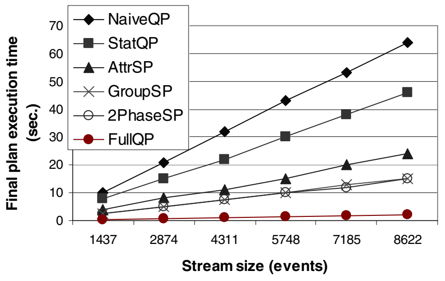
improvements in SQISLE can be made by making an execution plan compiler, which is likely to make the plan faster than C++ for selective queries.

In conclusion, query optimization, in particular runtime query optimization, improves performance substantially for selective queries. For selective queries the impact of query rewrites is relatively insignificant compared to query optimization.

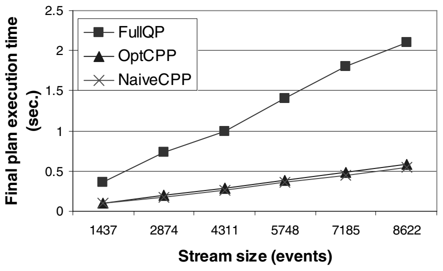
The loading approach in [9] took 15 seconds to load 25000 events. By contrast, the total processing in SQISLE with *FullQP* of the same number of events is 1.6s, which clearly shows the advantage with the streaming approach for our kind of applications.

In Table 2 the optimization strategies are compared by their *optimization overheads* obtained by subtracting the final plan execution time from the total query processing time. These overheads are independent of the stream size so the impact is negligible in practice for large streams. Table 2 contains performance measurements only for 25000 events, i.e. one file.

The optimization overhead of the ungrouped strategy *StatQP* is only the time to perform randomized optimization (29 seconds). The overhead of *AttrSP* (26 seconds) is dominated by the randomized optimization (80%). The remaining time is spent on collecting and monitoring statistics. The overheads of the grouped strategies (6.0 seconds for *GroupSP* and 4.5 seconds for *2PhaseSP*) are dominated (75%) by performing group profiling. To obtain the final execution plan *GroupSP* profiles only the first 40 events of the stream. The overhead of profiling all groups for a single event (0.15s) is substantial. The reason is that statistics are collected for all groups, including the very complex and expensive ones to get a good cost model. Therefore, it is



(a)



(b)

**Fig. 6.** Performance of different strategies for non-selective query *signalsixcuts*: (a) SQILSE strategies, (b) performance of the best strategy (*FullQP*) and the C++ implementations.

necessary to disable profiling once stream statistics are stabilized. Notice that overheads in both the ungrouped strategies are around four times higher than overheads of the grouped strategies, because the grouped strategies use the greedy optimization, which performs well, while for ungrouped strategies the greedy optimization did not produce good plans. Therefore the slow randomized optimization is used for ungrouped strategies.

Fig. 6(a) presents performance of the query processing strategies for the non-selective query *signalsixcuts* (16% events selected). The impact of the different query optimization strategies is less significant here. The best strategies (*GroupSP* and *2PhaseSP*) are just four times faster than the slowest (*NaiveQP*). Using the aggregate cost model (*StatQP*) gives a query plan that performs 28% better than *NaiveQP*. Using the attribute statistics profiling (*AttrSP*) gives a query plan that performs twice better than the query plan obtained without collecting statistics (*StatQP*). *GroupSP* and *2PhaseSP* are 35% faster than the *AttrSP*. The difference between *GroupSP* and *2PhaseSP* is again insignificant. We notice that query optimization has less impact on non-selective queries. In this case, the rewrites (*FullQP*) improve performance of the query by factor five compared to *2PhaseSP*. The other non-selective query *signalfourcuts* (58% events selected) performs similar to *signalsixcuts*.

Fig. 6(b) compares performance of the best query processing strategy (*FullQP*) with performance of the C++ implementations (*NaiveCPP* and *OptCPP*) for the non-selective query *signalsixcuts*. *FullQP* performs four times worse than both C++ implementations. The reason is that since the query is non-selective most operators are executed. Here, the cost of interpreting an operator in SQISLE is higher than the cost of executing machine instructions in C++, and we are comparing interpreted SQISLE

**Table 2.** Overhead times in seconds for query *bkgsixcuts* over events from one file

| Strategy | Total query processing time | Final plan execution time | Optimization overhead |
|---|---|---|---|
| StatQP | 253 | 224 | 29 |
| AttrSP | 136 | 110 | 26 |
| GroupSP | 9.4 | 1.9 | 7.5 |
| 2PhaseSP | 7.9 | 1.9 | 6.0 |
| FullQP | 6.1 | 1.6 | 4.5 |

with compiled C++. Implementing a compiler for query plans will reduce the interpretation overhead significantly.

In conclusion, query optimization, in particular runtime query optimization, improves performance significantly for all kinds of queries. For selective queries the improvements are dramatic. The impact of query rewrites is insignificant compared to query optimization of selective queries. For non-selective queries the combination of query optimization and query rewrite techniques significantly improves performance.

The evaluation demonstrates that query optimization techniques implemented in SQISLE can achieve performance for large and complex scientific queries close to a manually optimized C++ program.

## 5 Related Work

A visual query language for specifying HEP analyses is provided by the system PHEASANT [25]. HEP analyses are there defined in queries, which are translated into a general purpose programming language without any query optimization or simplification. By contrast, our system rewrites and optimizes queries, which is shown to give significant improvement in performance, approaching or surpassing that of hard-coded C++ programs.

Most developed DSMSs (e.g., Aurora [2], STREAM [1], TelegraphCQ [17], and XStream [10]) focus on infinite streams of rather simple objects and efficient processing of time-series operations over the streams, including stream aggregates and joins. The DSMSs are data driven and the continuous queries are rather simple. In contrast, in SQISLE the elements of the streams are complex objects (each object can be seen as a small database) and complex queries are applied on each streamed object independently from other objects. Thus the queries in SQISLE do not contain time-series operations and no join between streams is performed. Furthermore, SQISLE is demand driven, since it controls the stream flow.

In DBMSs, and in particular in DSMSs, precise statistics on data are not always available. Therefore, adaptive query processing (AQP) techniques are developed to improve query processing at query execution time by utilizing runtime feedback. AQP systems (e.g. [1,2,3,6,13,17,19,20]) continuously adapt the execution plan of a query to reflect significant changes in data statistics. By contrast SQISLE profiles a stream until statistical properties of the streamed objects are stabilized, and then reoptimizes the query using the stable statistics. This works well for our scientific applications where large numbers of complex objects having similar statistical properties (run conditions) are processed. After the statistics are stabilized, the rest of a stream is efficiently processed without profiling overhead.

Usually DSMSs (e.g. Aurora [2], STREAM [1], and TelegraphCQ [17]) schedule operators continuously per tuple and change the execution plan if significant flow changes are detected. Such monitoring for each simple data element adds significant overhead for large queries over complex objects. For Eddies [3] this overhead is even more significant, since optimization is performed whenever a tuple is scheduled for a next operator. To avoid the high cost of monitoring individual data elements, in SQISLE the profile-controller operator monitors the execution of an entire plan once per complex input stream object until sufficient statistics are collected about the objects, after which the plan is dynamically replaced with a final plan.

Some systems [6,13,19] generate several query execution plans and adaptively switch between them during query execution. Generating many execution plans during initial optimization is not feasible for large and complex queries. By contrast SQISLE generates only one initial query execution plan which is reoptimized at runtime using collected statistics to obtain a more efficient final execution plan.

The demand driven DBMS in [20] reoptimizes the entire query at runtime and then restarts the query based on already computed materialized intermediate results. SQISLE is also demand driven but need not restart the entire query since it reoptimizes only the processing query fragment that is applied on each subsequent streamed complex object produced by the static source access plan.

Query rewrites before cost-based query optimization has been demonstrated to improve performance for different kinds of applications in, e.g., engineering [28], image processing [21], and business processing [30]. SQISLE implements several rewrite rules and shows that they are particularly important for non-selective queries.

An example of implementing a complex scientific application in a DBMS with the loading approach is the Sloan Digital Sky Survey (SDSS) project [29]. In the project huge amounts of astronomical data from the SDSS telescope are loaded into a cluster of SQL Server databases and indexed. In our application efficient query specific indexes are required for calculating query dependent aggregated properties, e.g. based on number of isolated leptons, and static query independent indexing is not sufficient. Furthermore, the performance of first loading the data into a database and then processing them as queries is shown to be around ten times slower than processing the same data in one pass by SQISLE.

## 6 Summary and Future Work

The implementation was presented of a data stream management system SQISLE targeted to scientific applications where data are independent objects with complex structures selected by complex queries. SQISLE reads complex objects from files through a streamed wrapper interface and processes them in one pass efficiently by utilizing novel query processing techniques. Runtime query optimization methods collect stream statistics and reoptimize queries during execution. During query execution a *profile-controller* operator monitors collected statistics, reoptimizes the processing query fragment, and switches to another strategy, e.g. into non-profiled execution. Since the complex objects contain measurements produced in controlled experiments, we assume that statistical properties of complex objects, such as average number of different kinds of particles per event, produced in the same experiment are the same. Therefore profiling is performed only at the beginning of the one-pass data processing and then disabled to reduce profiling overhead.

To verify the approach, a scientific application from the ATLAS experiment [5,11] was implemented in SQISLE. The implementation demonstrated that performance of application analysis queries in SQISLE is close to a hard-coded and manually optimized C++ implementation of the same analysis, which requires a significant effort to develop.

In summary, the following techniques in SQISLE provide efficient processing of queries over streams of complex objects:

- The profile-controller operator enables more efficient execution plans for streamed queries than static cost-based query optimization, by choosing different query optimization strategies at runtime and then disabling the profiling.
- The query optimization techniques are shown to significantly improve performance of all kinds of queries.
- The query rewrite techniques are shown to improve performance significantly for non-selective queries, while being less effective for selective queries.

SQISLE currently interprets the generated query execution plans. By compiling the executions plans into C or machine code, the performance will be significantly better than the current implementation. Further improvements can be achieved by eliminating copying data from structures used in the ROOT files and structures used in SQISLE. It can be done either by storing collision event data in the ROOT files using data format used by SQISLE or by rewriting data management in SQISLE to operate on data having the same structure as in the ROOT files. Since the performance of SQISLE is already close to C++, these changes are likely to make SQISLE perform at least as well as a C++ program manually written by a physicist.

## References

1. Arasu, A., et al.: STREAM: The Stanford Stream Data Manager. IEEE Data Eng. Bull. 26, 19–26 (2003)
2. Abadi, D.J., et al.: Aurora: a new model and architecture for data stream management. VLDB J. 12, 120–139 (2003)
3. Avnur, R., Hellerstein, J.M.: Eddies: Continuously Adaptive Query Processing. In: SIGMOD Conference, pp. 261–272 (2000)
4. The ATLAS experiment, `http://atlasexperiment.org/`
5. Bisset, M., Moortgat, F., Moretti, S.: Trilepton+top signal from chargino-neutralino decays of MSSM charged Higgs bosons at the LHC. Eur. Phys. J. C 30, 419–434 (2003)
6. Babu, S., et al.: Adaptive Ordering of Pipelined Stream Filters. In: SIGMOD, pp. 407–418 (2004)
7. Brun, R., Rademakers, F.: ROOT - An Object Oriented Data Analysis Framework. In: AIHENP 1996 Workshop, Nucl. Inst. & Meth. in Phys. Res. A, vol. 389, pp. 81–86 (1997)
8. Fomkin, R.: Optimization and Execution of Complex Scientific Queries. Uppsala Dissertations from the Faculty of Science and Technology 80 (2009)
9. Fomkin, R., Risch, T.: Cost-based Optimization of Complex Scientific Queries. In: SSDBM, p. 1 (2007)
10. Girod, L., et al.: XStream: a Signal-Oriented Data Stream Management System. In: ICDE, pp. 1180–1189 (2008)
11. Hansen, C., Gollub, N., Assamagan, K., Ekelöf, T.: Discovery potential for a charged Higgs boson decaying in the chargino-neutralino channel of the ATLAS detector at the LHC. Eur. Phys. J. C 44, 1–9 (2005)
12. Ioannidis, Y.E., Christodoulakis, S.: On the Propagation of Errors in the Size of Join Results. In: SIGMOD, pp. 268–277 (1991)

13. Ives, Z.G., Halevy, A.Y., Weld, D.S.: Adapting to Source Properties in Processing Data Integration Queries. In: SIGMOD, pp. 395–406 (2004)
14. Ioannidis, Y.E., Kang, Y.C.: Randomized Algorithms for Optimizing Large Join Queries. In: SIGMOD, pp. 312–321 (1990)
15. Jones, N.D.: An Introduction to Partial Evaluation. ACM Comput. Surv. 28, 480–503 (1996)
16. Krishnamurthy, R., Boral, H., Zaniolo, C.: Optimization of Nonrecursive Queries. In: VLDB, pp. 128–137 (1986)
17. Krishnamurthy, S., et al.: TelegraphCQ: An Architectural Status Report. IEEE Data Eng. Bull. 26, 11–18 (2003)
18. Litwin, W., Risch, T.: Main Memory Oriented Optimization of OO Queries Using Typed Datalog with Foreign Predicates. IEEE Trans. Knowl. Data Eng. 4, 517–528 (1992)
19. Li, Q., et al.: Adaptively Reordering Joins during Query Execution. In: ICDE, pp. 26–35 (2007)
20. Markl, V., et al.: Robust Query Processing through Progressive Optimization. In: SIGMOD, pp. 659–670 (2004)
21. Marathe, A.P., Salem, K.: Query processing techniques for arrays. VLDB J. 11, 68–91 (2002)
22. Näs, J.: Randomized optimization of object oriented queries in a main memory database management system. Master's Thesis, `http://user.it.uu.se/~udbl/Theses/JoakimNasMSc.pdf`
23. Petrini, J.: Querying RDF Schema Views of Relational Databases. Uppsala Dissertations from the Faculty of Science and Technology 75 (2008)
24. Risch, T., Josifovski, V., Katchaounov, T.: Functional data integration in a distributed mediator system. In: Gray, P.M.D., Kerschberg, L., King, P.J.H., Poulovassilis, A. (eds.) The Functional Approach to Data Management: Modeling, Analyzing, and Integrating Heterogeneous Data. Springer, Heidelberg (2003)
25. Sousa, V., Amaral, V., Barroca, B.: Towards a full implementation of a robust solution of a domain specific visual query language for HEP physics analysis. In: J. Phys. Conf. Ser., vol. 119 (2007)
26. Selinger, P.G., et al.: Access Path Selection in a Relational Database Management System. In: SIGMOD, pp. 23–34 (1979)
27. Swami, A.N., Gupta, A.: Optimization of Large Join Queries. In: SIGMOD, pp. 8–17 (1988)
28. Sellis, T.K., Shapiro, L.D.: Query Optimization for Nontraditional Database Applications. IEEE Trans. Software Eng. 17, 77–86 (1991)
29. Szalay, A.S.: The Sloan Digital Sky Survey and beyond. SIGMOD Rec. 37, 61–66 (2008)
30. Vrhovnik, M., et al.: An Approach to Optimize Data Processing in Business Processes. In: VLDB, pp. 615–626 (2007)

# Comprehensive Optimization of Declarative Sensor Network Queries

Ixent Galpin, Christian Y.A. Brenninkmeijer, Farhana Jabeen, Alvaro A.A. Fernandes, and Norman W. Paton

University of Manchester, United Kingdom
{ixent,brenninc,jabeen,alvaro,norm}@cs.man.ac.uk

**Abstract.** We present a sensor network query processing architecture that covers all the query optimization phases that are required to map a declarative query to executable code. The architecture is founded on the view that a sensor network truly is a distributed computing infrastructure, albeit a very constrained one. As such, we address the problem of how to develop a comprehensive optimizer for an expressive declarative continuous query language over acquisitional streams as one of finding extensions to classical distributed query processing techniques that contend with the peculiarities of sensor networks as an environment for distributed computing.

## 1 Introduction

This paper addresses the problem of optimizing the evaluation of declarative queries over sensor networks (SNs) [1]. Throughout, by *sensor networks* we mean ad-hoc, wireless networks whose nodes are energy-constrained sensors endowed with general-purpose computing capability. We believe there is broad, continued interest [2,3,4] in exploring whether techniques developed in the context of classical database environments, such as query optimization, are also applicable and beneficial in non-classical ones, of which SNs are a comparatively recent example. Viewed as a distributed computing infrastructure, SNs are constrained to an unprecedented extent, and it is from such constraints that the challenges we have addressed arise. Addressing these challenges is important because of the prevailing expectations for the wide applicability of SNs [5]. These expectations imply that, as an infrastructural platform, SNs are bound to become more heterogeneous over time than previous work has assumed. Moreover, integrating SN data with data stemming from other networks (sensor-based or not) will be easier if the query processing technology used is less tied to specific execution environments.

We explore the hypothesis that the classical *two-phase optimization* approach [6] from distributed query processing (DQP) can be adapted to be effective and efficient over SNs, as initially proposed in [7]. Two-phase optimization is well established in the case of robust networks (e.g., the Internet, or the interconnect of a parallel machine), and involves the decomposition of query optimization into a *Single-Site* phase, and a subsequent *Multi-Site* phase, each of which is further decomposed into finer-grained decision-making steps. We aim to reuse well established optimization components where possible, and to identify steps that are necessarily different in SNs. We demonstrate that extending a classical DQP optimization architecture allows an expressive query language to be supported over resource-constrained networks, and that the resulting optimization steps yield good performance, as demonstrated through empirical evaluation.

**Related Work.** There have been many proposals in the so-called SN-as-database approach (including [2,8,3,4]). Surprisingly, none have fully described an approach to query optimization founded on a classical DQP architecture. Cougar papers [8] propose this idea but no publication describes its realization. SNQL [9] follows the idea through but no precise description (as provided by our algorithms) of the decision-making process has been published. Indeed, few publications provide systematic descriptions of complete query optimization architectures for SN query processors: the most comprehensive description found was for TinyDB [3], in which optimization is limited to operator reordering and the use of cost models to determine an appropriate acquisition rate given a user-specified lifetime. Arguably as a result of this, SN-as-database proposals have tended to limit the expressiveness of the query language. For example, TinyDB focuses on aggregation and provides limited support for joins. In many cases, assumptions are made that constrain the generality of the approach (e.g., Presto [2] focuses on storage-rich networks).

There has also been a tendency to address the optimization problem in a piecewise manner. For example, the use of probabilistic techniques to address the trade-off between acquiring data often and the cost of doing so is proposed in BBQ [10]; the trade-off between energy consumption and time-to-delivery is studied in WaveScheduling [11]; efficient and robust aggregation is the focus of several publications [12,13,14]; Bonfils [15] proposes a cost-based approach to adaptively placing a join which operates over distributed streams; Zadorozhny [16] uses an algebraic approach to generate schedules for the transmission of data in order to maximize the number of concurrent communications. However, these individual results are rarely presented as part of a fully-characterized optimization and evaluation infrastructure, giving rise to a situation in which research at the architecture level seems less well developed than that of techniques that might ultimately be applied within such query processing architectures.

This paper aims to provide a comprehensive, top-to-bottom approach to the optimization problem for expressive declarative continuous queries over potentially heterogeneous SNs. In comparison with past proposals, ours is broader, in that there are fewer compromises with respect to generality and expressiveness, and more holistic, in that it provides a top-to-bottom decomposition of the decision-making steps required to optimize a declarative query into a query execution plan (QEP).

**Approach and Main Results.** A key aspect of our approach is that it supports the optimization and evaluation of SNEEql (for Sensor NEtwork Engine query language), a comprehensive stream query language, inspired by classical stream languages such as CQL [17]. Thus, we do not start from the assumption that a query language for use with resource-constrained devices must, as a consequence, provide limited functionality. We then take a classical DQP optimizer architecture as a starting point, and adapt it to consider how the optimization and evaluation of queries for SNEEql differs if, rather than targeting classical distributed computational resources, we target SNs. We identify what assumptions in classical DQP are violated in the SN case, and propagate the consequences of such violations into the design and engineering of a DQP architecture for SN data management. The differences we consider here that have led to adaptations and extensions, are: (i) *the acquisitional nature of the query processing task*: data is neither lying ready in stores nor is it pushed (as in classical streams), but requested; (ii) *the energy-constrained nature of the sensors*: preserving energy becomes a crucial requirement because it is a major determinant of network longevity, and requires the executing code to shun energy-hungry tasks; (iii) *the fundamentally different nature of the communication links*: wireless links are not robust, and often cannot span the desired distances, so the data flow topology (e.g., as embodied in a query operator tree) needs to be overlaid onto some query-specific network topology (e.g., a routing tree of radio-level links) for data to flow from sensors to clients, and the two trees are not isomorphic (e.g., some network nodes act as pure relay nodes); and (iv) *the need to run sensor nodes according to data-dependent duty cycles*: each element in the computational fabric must act in accordance with an agenda that co-ordinates its activity with that of other elements on which it is dependent or that depend on it, thereby enabling energy management (e.g., by sending devices to energy-saving states until the next activity). We note that these points also distinguish our approach from infrastructures for stream query processing (e.g., [17]), which do not operate in resource constrained environments.

**Summary of Contributions.** The body of the paper describes: (1) a user-level syntax (Section 2) and algebra (Section 3.1) for SNEEql, an expressive language for querying over acquisitional sensor streams; (2) an architecture for the optimization of SNEEql, building on well-established DQP components where possible, but making enhancements or refinements where necessary to accommodate the SN context (Section 3); (3) algorithms that instantiate the components, thereby supporting integrated query planning that includes routing, placement and timing (Section 3.2); and (4) an evaluation of the resulting infrastructure, demonstrating the impact of design and optimization decisions on query performance, and the increased empowering to the user (Section 4) who is able to trade-off different Qualities-of-Service (Qos) according to application needs.

## 2 Query Language

SNEEql [18] is a declarative query language for SNs inspired by expressive classical stream query languages such as CQL [17]. A rich language is used even

```
Schema:  outflow (id, time, temp, turbidity, pressure)          sources: {0, 2, 4}
         inflow  (id, time, temp, pressure, ph)                 sources: {4, 5, 7}

Q1:  SELECT RSTREAM id, pressure                    Q2:  SELECT RSTREAM AVG(pressure)
     FROM   inflow[NOW]                                  FROM   inflow[NOW]
     WHERE  pressure > 500;

Q3:  SELECT RSTREAM o.id, i.id, o.time, o.pressure, i.pressure
     FROM   outflow[NOW] o, inflow[FROM NOW - 1 TO NOW - 1 MINUTES] i
     WHERE  o.pressure < i.pressure AND i.pressure > 500

QoS: {ACQUISITION RATE = 3s ; DELIVERY TIME = 5s}
```

**Fig. 1.** Schema Metadata, Example queries in SNEEql and QoS Expectations

though our target delivery platform consists of limited devices because: (i) the results of queries written using inexpressive query languages may require offline post-processing over data that has to be delivered using costly wireless communication; and (ii) sensor applications require comprehensive facilities for correlating data sensed in different locations at different times (e.g., [19,20]).

An example schema and queries are given in Fig. 1 motivated by the application scenario (but not actually occurring) in PipeNet, "a system based on wireless SNs [that] aims to detect, localize and quantify bursts and leaks and other anomalies in water transmission pipelines" [20]. The *logical extents* in the schema (i.e., `inflow` and `outflow`) comprise a (possibly overlapping) subset of the acquisitional streams generated by the source nodes, as specified in Fig. 1. The topology of the network is depicted in Fig. 2. *Q1* requests the tuples from `inflow` whose pressure attribute is above a certain threshold; *Q2* requests the average value of the pressure readings in the `inflow` logical extent; and *Q3* obtains instances in which the `outflow` pressure is less than the `inflow` pressure a minute before (as long as the latter was above a certain threshold). All acquire data and return results every 3 seconds, as stated in the QoS parameters. *Q3* returns the `id` corresponding to the `inflow` and `outflow` source nodes, to assist the user with locating a potential leak. The examples illustrate how SNEEql can express select-project (*Q1*), aggregation (*Q2*) and join (*Q3*) queries. *Q3* is noteworthy as it correlates data from different locations at different times, and cannot be expressed with previous SN query languages.

In SNEEql, the only structured type is tuple. The primitive collection types in SNEEql are: relation, an instance of which is a bag of tuples with definite cardinality; window, an instance of which is a relation whose content may implicitly vary between evaluation episodes; and stream, an instance of which is a bag of tuples with indefinite cardinality whose content may implicitly vary throughout query evaluation. As in CQL, operations construct windows out of streams and vice-versa. In all the queries, windows are used to convert from streams to relations, relational operators act on those relations, and stream operators add the resulting tuples into the output stream. Window definitions are of the form *WindowDimension* [SLIDE] [*Units*], where the *WindowDimension* is of the form NOW or FROM *Start* TO *End*, where the former contains all the tuples with the current time stamp, and the latter contains all the tuples that

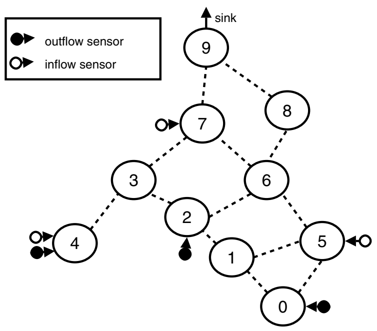


**Fig. 2.** Connectivities and Modalities

fall within the given range. The *Start* and *End* of a range are of the form `NOW` or `NOW` –*Literal*, where the *Literal* represents some number of *Units*, which is either `ROWS` or a time unit (`HOURS`, `MINUTES` or `SECONDS`). The optional `SLIDE` indicates the gap in *Units* between the *Start* of successive windows; this defaults to the acquisition rate specified. The results of relational operators are added to the result stream using the relation-to-stream operators from CQL, namely `ISTREAM`, `DSTREAM` and `RSTREAM`, denoting *inserted-only*, *deleted-only* and *all-tuples*, respectively.

In a stream query language, where conceptually data is being consumed, and thus potentially produced, on an ongoing basis, the question exists as to when a query should be evaluated. In SNEEql, an *evaluation episode* is determined by the acquisition of data, i.e., setting an acquisition rate sets the rate at which evaluation episodes occur. Thus, whenever new data is acquired, it is possible that new results can be derived. In the implementation, however, partial query results may be cached within the network with a view to minimizing network traffic, and different parts of a query may be evaluated at different times, reflecting related, but distinct, QoS expectations, viz., the *acquisition rate* and the *delivery time*, as shown in Fig. 1. Noteworthy features of SNEEql illustrated in Fig. 1 include: (i) extending CQL stream-to-window capabilities to allow for windows whose endpoint is earlier than now or than the last tuple seen, as happens in *Q3* with the window on `inflow`, which emits tuples that were acquired one minute ago; (ii) allowing sensing capabilities to be logically abstracted in a schema (like Cougar [8], but unlike TinyDB, which assumes that all sensed data comes from a single extent denoted by the keyword `SENSORS`); (iii) allowing the logical streams to stem from more than one set of sensor nodes, and possibly intersecting ones (as is the case in Fig. 1); (iv) expressing joins, where the tuples come from different locations in the SN, as a single query and without using materialization points (as would be required in TinyDB); and (v) allowing QoS expectations to be set for the optimizer, such as acquisition rate and delivery time. A detailed description of the SNEEql language, including a formal semantics, is given in [18].

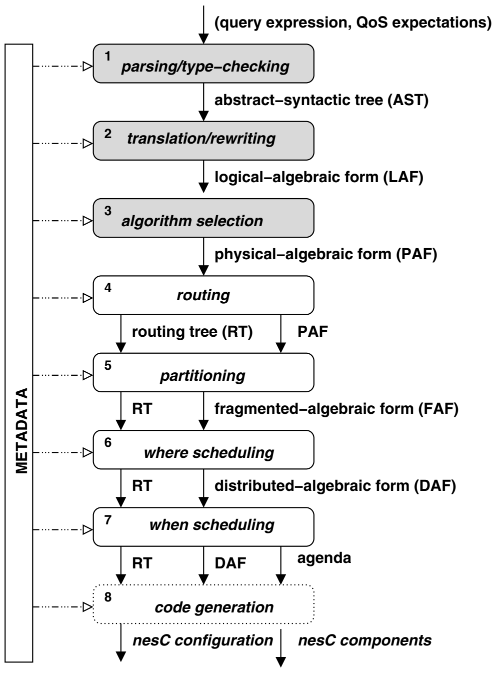


**Fig. 3.** SNEEql Compiler/Optimizer Stack

## 3 Query Compiler/Optimizer

Recall that our goal is to explore the hypothesis that extensions to a classical DQP optimization architecture can provide effective and efficient query processing over SNs. The SNEEql compilation/optimization stack is illustrated in Fig. 3, and comprises three phases. The first two are similar to those familiar from the two phase-optimization approach, namely *Single-Site* (comprising Steps 1-3, in gray boxes) and *Multi-Site* (comprising steps 4-7, in white, solid boxes). The *Code Generation* phase grounds the execution on the concrete software and hardware platforms available in the network/computing fabric and is performed in a single step, Step 8 (in a white, dashed box), which generates executable code for each site based on the distributed QEP, routing tree and agenda. The optimizer consists of 15K lines of Java.

Classical kinds of metadata (e.g., schematic information, statistics, etc.) are used by the SNEEql optimizer, but we focus here on the requirement for a more detailed description of the network fabric. Thus, Fig. 2 depicts an example network for the case study in Fig. 1 in terms of a weighted connectivity graph and the sensing modalities available in each site. Dotted edges denote that single-hop communication is possible. Weights normally model energy, but more complex weights could be used. Here, we assume links to have unit costs, but this need not be so. Note that all sites may contribute computing (e.g., processing intermediate results) or communication (e.g., relaying data) capabilities. Metadata is assumed to be populated by a network management layer, such as Moteworks (`http://www.xbow.com`), which also provides services such as network formation, self-healing, code dissemination etc. Note, however, we have not actually used Moteworks since it does not support simulation, and we have opted to simulate in order to efficiently carry out systematic experimentation.

Throughout the steps in the query stack, size-, memory-, energy- and time-cost models are used by the optimizer to reach motivated decisions. These cannot be described in detail due to space limitations. The cost models are very close in style and level of detail to those in [21], but have been extended and adapted for the SN context.

## 3.1 Phase 1: Single-Site Optimization

Single-site optimization is decomposed into components that are familiar from classical, centralized query optimizers. We make no specific claims regarding the novelty of these steps, since the techniques used to implement them are well-established. In essence: *Step 1* checks the validity of the query with respect to syntax and the use of types, and builds an abstract syntax tree to represent the query; *Step 2* translates the abstract syntax tree into a logical algebra, the operators of which are reordered to reduce the size of intermediate results; and *Step 3* translates the logical algebra into a physical algebra, which, e.g., makes explicit the algorithms used to implement the operators. Fig. 6 depicts the outcome of Steps 1 to 3 for *Q3* from Fig. 1, expressed using SNEEql *physical-algebraic form* (PAF) that is the principal input to multi-site optimization.

Table 1 describes the PAF operators, grouped by their respective input-to-output collection types. A signature has the form OPERATOR_NAME[Parameters](InputArgumentTypes):OutputArgumentTypes, where the argument types are denoted $R$, $S$ and $W$, for relation, stream and window respectively, and a vertical bar indicates a choice of one of the types given. ACQUIRE and DELIVER denote data sources and sinks, respectively. The window on `inflow` is represented in the algebra in milliseconds as TIME_WINDOW[t-60000,t-60000,30000], and is relative to `t`, which is bound in turn to the time in which each evaluation episode of the query starts. Note that the ACQUIRE and DELIVER are both *location sensitive*, i.e., there is no leeway as to which node(s) in the SN they may execute on. Furthermore, NL_JOIN is *attribute sensitive*, i.e., in order to carry out partitioned-parallelism the optimizer needs to consider how the input(s) are partitioned in order to preserve operator semantics.

## 3.2 Phase 2: Multi-site Optimization

For distributed execution, the physical-algebraic form (PAF) is broken up into QEP fragments for evaluation on specific nodes in the network. In a SN, consideration must also be given to routing (the means by which data travels between nodes within the network) and duty cycling (when nodes transition from being switched on and engaged in specific tasks, and being asleep, or in power-saving

**Table 1.** SNEEql Physical Algebra

| | |
|---|---|
| Stream-to-Stream Operators | |
| ACQUIRE[AcquisitionInterval, PredExpr, AttrList]$(S)$ : $S$ | Take sensor readings every AcquisitionInterval for sources in $S$ and apply SELECT[PredExpr] and PROJECT[AttrList]. *LocSen*. |
| DELIVER[ ]$(S)$ : $S$ | Deliver the query results. *LocSen*. |
| Stream-to-Window Operators | |
| TIME_WINDOW[startTime, endTime, Slide]$(S)$ : $W$ | Define a time-based window on $S$ from startTime to endTime inclusive and re-evaluate every slide time units. |
| ROW_WINDOW[startRow, endRow, Slide]$(S)$ : $W$ | Define a tuple-based window on $S$ from startRow to endRow inclusive and re-evaluate every slide rows. *AttrSen*. |
| Window-to-Stream Operators | |
| RSTREAM[ ]$(W)$ : $S$ | Emit all the tuples in $W$. |
| ISTREAM[ ]$(W)$ : $S$ | Emit the newly-inserted tuples in $W$ since the previous window evaluation. |
| DSTREAM[ ]$(W)$ : $S$ | Emit the newly-deleted tuples in $W$ since the previous window evaluation. |
| Window-to-Window or Relation-to-Relation Operators | |
| NL_JOIN[ProjectList, PredExpr]$(R\|W,R\|W)$ : $R\|W$ | Join tuples on PredExpr condition using nested-loop algorithm. *AttrSen*. |
| AGGR_INIT[AggrFunction, AttrList]$(R\|W)$ : $R\|W$ | Initialize incremental aggregation for attributes in AttrList for type of aggregation specified by AggrFunction. *AttrSen*. |
| AGGR_MERGE[AggrFunction, AttrList]$(R\|W)$ : $R\|W$ | Merge partial-result tuples of incremental aggregation for attributes in AttrList for type of aggregation specified by AggrFunction. *AttrSen*. |
| AGGR_EVAL[AggrFunction, AttrList]$(R\|W)$ : $R\|W$ | Evaluate final result of incremental aggregation for attributes in AttrList for type of aggregation specified by AggrFunction. *AttrSen*. |
| Any-to-Same-as-input-type Operators | |
| SELECT[PredExpr]$(R\|S\|W)$: $R\|S\|W$ | Eliminate tuples which do not meet PredExpr predicate. |
| PROJECT[AttrList] $(R\|S\|W)$: $R\|S\|W$ | Generate tuple with AttrList attributes. |

modes). Therefore, for Steps 4-7, we consider the case of robust networks and the contrasting case of SNs.

For execution over multiple nodes in robust networks, the second phase is comparatively simple: one step partitions the PAF into fragments and another step allocates them to suitably resourced sites, as in, e.g., [22]. One approach to achieving this is to map the physical-algebraic form of a query to a distributed one in which EXCHANGE operators [23] define boundaries between fragments. An EXCHANGE operator encapsulates all of control flow, data distribution and inter-process communication and is implemented in two parts, referred to as producer and consumer. The former is the root operator of the upstream fragment, and the latter, a leaf operator of the downstream one. This approach has been successful in DQP engines for the Grid that we developed in previous work [24,25].

However, for the same general approach to be effective and efficient in a SN, a response is needed to the fact that assumptions that are natural in the robust-network setting cease to hold in the new setting and give rise to a different set of challenges, the most important among which are the following: **C1**: *location and time are both concrete*: acquisitional query processing is grounded on the physical world, so sources are located and timed in concrete space and time, and the optimizer may need to respond to the underlying geometry and to synchronization issues; **C2**: *resources are severely bounded*: sensor nodes can be depleted of energy, which may, in turn, render the network useless; **C3**: *communication events are overly expensive*: they have energy unit costs that are typically an order of magnitude larger than the comparable cost for computing and sensing events; and **C4**: *there is a high cost in keeping nodes active for long periods*: because of the need to conserve energy, sensor node components must run tight duty cycles (e.g., going to sleep as soon they become idle).

Our response to this different set of circumstances is reflected in Steps 4-7 in Fig. 3, where rather than a simple partition-then-allocate approach (in which a QEP is first partitioned into fragments, and these fragments are then allocated to specific nodes on the network), we: (a) introduce Step 4, in which the optimizer determines a routing tree for communication links that the data flows in the operator tree can then rely on, with the aim of addressing the issue that paths used by data flows in a query plan can greatly impact energy consumption (a consequence of **C3**); (b) preserve the query plan partitioning step, albeit with different decision criteria, which reflect issues raised by **C1**; (c) preserve the scheduling step (which we rename to *where-scheduling*, to distinguish it from Step 7), in which the decision is taken as to where to place fragment instances in concretely-located sites (e.g., some costs may depend on the geometry of the SN, a consequence of **C1**); and (d) introduce *when-scheduling*, the decision as to when, in concrete time, a fragment instance placed at a site is to be evaluated (and queries being continuous, there are typically many such episodes) to address **C1** and **C4**. **C2** is taken into account in changes throughout the multi-site phase.

For each of the following subsections that describe Steps 4 to 7, we indicate how the proposed technique relates to DQP and to TinyDB, the former because we have used established DQP architectures as our starting point, and the latter because it is the most fully characterized proposal for a SN query processing system. The following notation is used throughout the remainder of this section. Given a query $Q$, let $P_Q$ denote the graph-representation of the query in physical-algebraic form. Throughout, we assume that: (1) operators (and fragments) are described by properties whose values can be obtained by traditional accessor functions written in dot notation (e.g., $P_Q$.Sources returns the set of sources in $P_Q$); and (2) the data structures we use (e.g., sets, graphs, tuples) have functions with intuitive semantics defined on them, written in applicative notation (e.g., for a set $S$, ChooseOne($S$) returns any $s \in S$; for a graph $G$, EdgesIn($G$) returns the edges in $G$); Insert(($v_1$,$v_2$),$G$) inserts the edge ($v_1$,$v_2$) in $G$.

**Routing.** Step 4 in Fig. 3 decides which sites to use for routing the tuples involved in evaluating $P_Q$. The aim is to generate a routing tree for $P_Q$ which is economical with respect to the total energy cost required to transmit tuples. Let $G = (V, E, d)$ be the weighted connectivity graph for the target SN (e.g., the one in Fig. 2). Let $P_Q$.Sources $\subseteq G.V$ and $P_Q$.Destination $\in G.V$ denote, respectively, the set of sites that are data sources, and the destination site, in $P_Q$. The aim is, for each source site, to reduce the total cost to the destination. We observe that this is an instance of the *Steiner tree* problem, in which, given a graph, a tree of minimal cost is derived which connects a required set of nodes (the *Steiner nodes*) using any additional nodes which are necessary [1]. Thus, the SNEEql-optimal routing tree $R_Q$ for $Q$ is the Steiner tree for $G$ with Steiner nodes $P_Q$.Sources $\cup$ $\{P_Q$.Destination$\}$. The problem of computing a Steiner tree is NP-complete, so the heuristic algorithm given in [1] (and reputed to perform well in practice) is used to compute an approximation. The resulting routing tree for $Q3$ over the network given in Fig. 2 is depicted in Fig. 7.

*Relationship to DQP:* The routing step has been introduced in the SN context due to the implications of the high cost of wireless communications, viz., that the paths used to route data between fragments in a query plan have a significant bearing on its cost. Traditionally, in DQP, the paths for communication are solely the concern of the network layer. In a sense, for SNEEql, this is also a preparatory step to assist *where-scheduling* step, in that the routing tree imposes constraints on the data flows, and thus on where operations can be placed.

*Relationship to Related Work:* In TinyDB, routing tree formation is undertaken by a distributed, parent-selection protocol at runtime. Our approach aims, given the sites where location-sensitive operators need to be placed, to reduce the distance traveled by tuples. TinyDB does not directly consider the locations of data sources while forming its routing tree, whereas the approach taken here makes finer-grained decisions about which depletable resources (e.g., energy) to make use of in a query. This is useful, e.g., if energy stocks are consumed at different rates at different nodes.

**Partitioning.** Step 5 in Fig. 3 defines the fragmented form $F_Q$ of $P_Q$ by breaking up selected edges $(child, op) \in P_Q$ into a path $[(child, e_p), (e_c, op)]$ where $e_p$ and $e_c$ denote, respectively, the producer and consumer parts of an EXCHANGE operator. The edge selection criteria are semantic, in the case of location- or attribute-sensitive operators in which correctness criteria constrain placement, and pragmatic in the case of an operator whose output size is larger than that of its child(ren) in which case placement seeks to reduce overall network traffic. Let Size estimate the size of the output of an operator or fragment, or the total output size of a collection of operator or fragment siblings. The algorithm that computes $F_Q$ is shown in Fig. 4. Fig. 8 depicts the distributed-algebraic form (the output of where-scheduling) given the routing tree in Fig. 7 for the physical-algebraic form in Fig. 6. The EXCHANGE operators that define the four fragments shown in Fig. 8 are placed by this step. The fragment identifier `Fn` denotes the fragment number. The assigned set of sites for each fragment (below the fragment identifier) are determined subsequently in where-scheduling.

```
FRAGMENT-DEFINITION($P_Q$, Size)
1  $F_Q \leftarrow P_Q$
2  while   $\triangleright$ post-order traversing $F_Q$,
           $\triangleright$ let $op$ denote the current operator
3      do for each $child \in op$.Children
4             do if Size($op$) > Size($op$.Children) or $op$.LocationSensitive = yes
5                or $op$.AttributeSensitive = yes
6                  then Delete(($child$, $op$), $P_Q$) ; Insert(($child$, $e_p$), $P_Q$)
7                       Insert(($e_p$, $e_c$), $P_Q$) ; Insert(($e_c$, $op$), $P_Q$)
8  return $F_Q$
```

**Fig. 4.** The partitioning algorithm

EXCHANGE has been inserted between each ACQUIRE and the JOIN, because the predicate of the latter involves tuples from different sites, and therefore data redistribution is required. Note also that an EXCHANGE has been inserted below the DELIVER, because the latter is (as is ACQUIRE) *location sensitive*, i.e., there is no leeway as to where it may be placed.

*Relationship to DQP:* This step differs slightly from its counterpart in DQP. EXCHANGE operators are inserted more liberally at edges where a reduction in data flow will occur, so that radio transmissions take place along such edges whenever possible.

*Relationship to Related Work:* Unlike SNEEql/DQP, TinyDB does not partition its query plans into fragments. The entire query plan is shipped to sites which are required to participate in it, even if they are just relaying data.

**Where-Scheduling.** Step 6 in Fig. 3 decides which QEP fragments are to run on which routing tree nodes. This results in the distributed-algebraic form of the query. Creation and placement of fragment instances is mostly determined by semantic constraints that arise from location sensitivity (in the case of ACQUIRE and DELIVER operators) and attribute sensitivity (in the case JOIN and aggregation operators, where tuples in the same logical extent may be traveling through different sites in the routing tree). Provided that location and attribute sensitivity are respected, the approach aims to assign fragment instances to sites, where a reduction in result size is predicted (so as to be economical with respect to the radio traffic generated).

Let $G$, $P_Q$ and $F_Q$ be as above. Let $R_Q$ = ROUTING($P_Q$, $G$) be the routing tree computed for $Q$. The where-scheduling algorithm computes $D_Q$, i.e., the graph-representation of the query in distributed-algebraic form, by deciding on the creation and assignment of fragment instances in $F_Q$ to sites in the routing tree $R_Q$. If the size of the output of a fragment is smaller than that of its child(ren) then it is assigned to the deepest possible site(s) (i.e., the one with the longest path to the root) in $R_Q$, otherwise it is assigned to the shallowest site for which there is available memory, ideally the root. The aim is to reduce radio traffic (by postponing the need to transmit the result with increased size). Semantic criteria dictate that if a fragment contains a location-sensitive operator, then instances of it are created and assigned to each corresponding site (i.e., one that acts as source or destination in $F_Q$). Semantic criteria also dictate that if a fragment contains an attribute-sensitive operator, then an instance of it is created and assigned to what we refer to as a confluence site for the operator.

To grasp the notion of a *confluence site* in this context, note that the extent of one logical flow (i.e., the output of a logical operator) may comprise tuples that, in the routing tree, travel along different routes (because, ultimately, there may be more than one sensor feeding tuples into the same logical extent). In response to this, instances of the same fragment are created in different sites, in which case EXCHANGE operators take on the responsibility for data distribution among fragment instances (concomitantly with their responsibility for mediating communication events). It follows that a fragment instance containing an attribute-sensitive operator is said to be effectively-placed only at sites in which the logical extent of its operand(s) has been reconstituted by confluence. Such sites are referred to as confluence sites. For a JOIN, a confluence site is a site through which all tuples from both its operands travel. In the case of aggregation operators, which are broken up into three physical operators (viz., AGGR_INIT, AGGR_MERGE, AGGR_EVAL), the notion of a confluence site does not apply to an AGGR_INIT. For a binary AGGR_MERGE (such as for an AVG, where AGGR_MERGE updates a (SUM, COUNT) pair), a confluence site is a site that tuples from both its operands travel through. Finally, for an AGGR_EVAL, a confluence site is a site through which tuples from all corresponding AGGR_MERGE operators travel. The most efficient confluence site to which to assign a fragment instance is considered to be the deepest, as it is the earliest to be reached in the path to the destination and hence the most likely to reduce downstream traffic.

Let $P_Q$ and $R_Q$ be as above. Let $s \mathit{\Delta}\ op$ be true iff $s$ is the deepest confluence site for $op$. The algorithm that computes $D_Q$ is shown in Fig. 5. The resulting $D_Q$ for the example query is shown in Fig. 8. It can be observed that instances of F2 and F3 have been created at multiple sites, as these fragments contain location-sensitive ACQUIRE operators, whose placement is dictated by the schema definition in Fig. 1. Also, a single instance of attribute-sensitive F1 has been created and assigned to site 7, the deepest confluence site where tuples from both F2 and F3 are available (as it is a non-location-sensitive fragment and has been placed according to its expected output size, to reduce communication). Note also the absence of site 3 in Fig. 8 wrt. Fig. 7. This is because site 3 is only a relay node in the routing tree.

*Relationship to DQP:* Compared to DQP, here the allocation of fragments is constrained by the routing tree, and operator confluence constraints, which enables the optimizer to make well-informed decisions (based on network topology) about where to carry out work. In classical DQP, the optimizer does not have to consider the network topology, as this is abstracted away by the network protocols. As such, the corresponding focus of where-scheduling in DQP tends to be on finding sites with adequate resources (e.g., memory and bandwidth) available to provide the best response time (e.g., Mariposa [22]).

*Relationship to Related Work:* Our approach differs from that of TinyDB, since its QEP is never fragmented. In TinyDB, a node in the routing tree either

```
FRAGMENT-INSTANCE-ASSIGNMENT($F_Q$, $R_Q$, Size)
 1  $D_Q \leftarrow F_Q$
 2  while    ▷ post-order traversing $D_Q$
             ▷ let $f$ denote the current fragment
 3      do if $op \in f$ and $op$.LocationSensitive = yes
 4          then for each $s \in op$.Sites
 5                  do Assign($f$.New, $s$, $D_Q$)
 6         elseif $op \in f$ and $op$.AttributeSensitive = yes
 7          andSize($f$) < Size($f$.Children)
 8          then while    ▷ post-order traversing $R_Q$,
                          ▷ let $s$ denote the current site
 9                  do if $s \Delta op$
10                      then Assign($f$.New, $s$, $D_Q$)
11         elseif Size($f$) < Size($f$.Children)
12          then for each $c \in f$.Children
13                  do for each $s \in c$.Sites
14                          do Assign($f$.New, $s$, $D_Q$)
15          else Assign($f$.New, $R_Q$.Root, $D_Q$)
16  return $D_Q$
```

**Fig. 5.** The *where-scheduling* algorithm

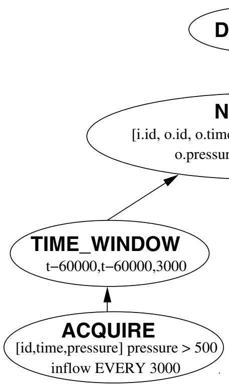


**Fig. 6.** *Q3* Physical-Algebraic Form

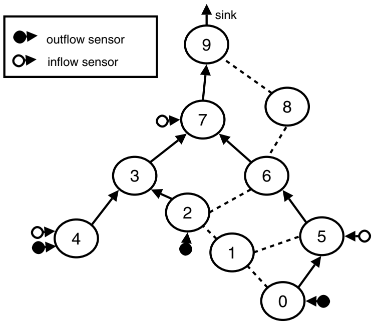


**Fig. 7.** *Q3* Routing Tree

(i) evaluates the QEP, if the site has data sources applicable to the query, or (ii) restricts itself to relaying results to its parent from any child nodes that are evaluating the QEP. Our approach allows different, more specific workloads to be placed in different nodes. For example, unlike TinyDB, it is possible to compare results from different sites in a single query, as in Fig. 8. Furthermore, it is also possible to schedule different parts of the QEP to different sites on the basis of the resources (memory, energy or processing time) available at each site. The SNEEql optimizer, therefore, responds to resource heterogeneity in the fabric. TinyDB responds to excessive workload by shedding tuples, replicating the strategy of stream processors (e.g., STREAM [17]). However, in SNs, since there is a high cost associated with transmitting tuples, load shedding is an undesirable option. As the query processor has control over data acquisition, it

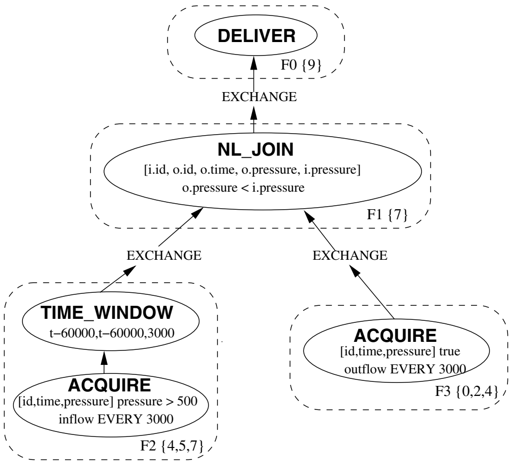


**Fig. 8.** *Q3* Distributed-Algebraic Form

| Time (ms) | Sites | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 5 | 6 | 2 | 4 | 3 | 7 | 9 |
| 0 | $F3_1$ | $F2_1$ | | $F3_1$ | $F2_1$ | | $F2_1$ | |
| 63 | | | | | $F3_1$ | | | |
| 3000 | $F3_2$ | $F2_2$ | | $F3_2$ | $F2_2$ | | $F2_2$ | |
| 3032 | | | | *tx3* | | *rx2* | | |
| 3063 | *tx5* | *rx0* | | | $F3_2$ | | | |
| 3125 | | | | | *tx3* | *rx4* | | |
| 3157 | | *tx6* | *rx5* | | | | | |
| 3313 | | | *tx7* | | | | *rx6* | |
| 3469 | | | | | | *tx7* | *rx3* | |
| 3688 | | | | | | | F1 | |
| 3719 | | | | | | | *tx9* | *rx7* |
| 4313 | | | | | | | | F0 |
| 4344 | | | | | | | | |

**Fig. 9.** *Q3 Agenda*

seems more appropriate to tailor the optimization process so as to select plans that do not generate excess tuples in the first place.

**When-Scheduling.** Step 7 in Fig. 3 stipulates execution times for each fragment. Doing so efficiently is seldom a specific optimization goal in classical DQP. However, in SNs, the need to co-ordinate transmission and reception and to abide by severe energy constraints make it important to favor duty cycles in which the hardware spends most of its time in energy-saving states. The approach adopted by the SNEEql compiler/optimizer to decide on the timed execution of each fragment instance at each site is to build an agenda that, insofar as permitted by the memory available at the site, and given the acquisition rate $\alpha$ and the delivery time $\delta$ set for the query, buffers as many results as possible before transmitting. The aim is to be economical with respect to both the time in which a site needs to be active and the amount of radio traffic that is generated.

The agenda is built by an iterative process of adjustment. Given the memory available at, and the memory requirements of the fragment instances assigned to, each site, a candidate buffering factor $\beta$ is computed for each site. This candidate $\beta$ is used, along with the acquisition rate $\alpha$, to compute a candidate agenda. If the candidate agenda *makespan* (i.e., the time that the execution of the last task is due to be completed) exceeds the smallest of the delivery time $\delta$ and the product of $\alpha$ and $\beta$, the buffering factor is adjusted downwards and a new candidate agenda is computed. The process stops when the makespan meets the above criteria. Let Memory, and Time, be, respectively, a model to estimate the memory required by, and the execution time of, an operator or fragment. The algorithm that computes the agenda is shown in Fig. 10 and 11.

The agenda can be conceptualized as a matrix, in which the rows, identified by a relative time point, denote concurrent tasks in the sites which identify the columns. For Fig. 8, the computed agenda is shown in Fig. 9, where $\alpha = 3000$ms,

```
WHEN-SCHEDULING($D_Q$, $R_Q$, $\alpha$, $\delta$, Memory, Time)
 1  while      ▷ pre-order traversing $R_Q$,
               ▷ let $s$ denote the current site
 2        do $reqMem_e \leftarrow reqMem_f \leftarrow 0$
 3           for each $f \in s$.AssignedFragments
 4               do $x \leftarrow$ Memory($f$.EXCHANGE)
 5                  $reqMem_f \leftarrow +$ Memory($f$) - $x$
 6                  $reqMem_e \leftarrow + x$
 7           $\beta^*[s] \leftarrow \lfloor \frac{s.\mathsf{AvailableMemory} - reqMem_f}{reqMem_e} \rfloor$
 8  $\beta \leftarrow$ min($\beta^*$)
 9  while $agenda$.Makespan $>$ min($\alpha * \beta, \delta$)
10        do $agenda \leftarrow$ BUILD-AGENDA($D_Q$, $R_Q$, $\alpha$, $\beta$, Time)
11           decr($\beta$)
12  return $agenda$
```

**Fig. 10.** Computing a SNEEql Execution Schedule

$\beta = 2$ and $\delta = 5000$ms. Thus, a non-empty cell $(t, s)$ with value $a$, denotes that task $a$ starts at time $t$ in site $s$. In an agenda, there is a column for each site and a row for each time when some task is started. Thus, if cell $(t, s) = a$, then at time $t$ in site $s$, task $a$ is started. A task is either the evaluation of a fragment (which subsumes sensing), denoted by `Fn` in Fig. 9, where `n` is the fragment number, or a communication event, denoted by $tx\ n$ or $rx\ n$, i.e., respectively, tuple transmission to, or tuple reception from, site $n$. Note that leaf fragments `F2` and `F3` are annotated with a subscript, as they are evaluated $\beta$ times in each agenda evaluation. Blank cells denote the lack of a task to be performed at that time for the site, in which case, an OS-level power management component is delegated the task of deciding whether to enter a energy-saving state.

In SNEEql (unlike TinyDB), tuples from more than one evaluation time can be transmitted in a single communication burst, thus enabling the radio to be switched on for less time, and also saving the energy required to power it up and down. This requires tuples between between evaluations to be buffered, and results in an increase in the time-to-delivery. Therefore, the buffering factor is constrained by both the available memory and by user expectations as to the delivery time. Note that, query evaluation being continuous, the agenda repeats. The period with which it does so is $p = \alpha\beta$, i.e., $p = 3000 * 2 = 6000$ for the example query. Thus, the acquisition rate $\alpha$ dictates when an ACQUIRE executes; $\alpha$ and the buffering factor $\beta$ dictate when a DELIVER executes. In this example, note that the agenda makespan is $4344ms$. This is calculated by summing the duration of tasks in the agenda (taking into account whether each task has been scheduled sequentially, or concomitantly, in relation to other tasks). Therefore, the delivery time specified in Fig. 1 is met by the example agenda.

*Relationship to DQP:* The time-sensitive nature of data acquisition in SNs, the delivery time requirements which may be expressed by the user, the need for wireless communications to be co-ordinated and for sensor nodes to duty-cycle, all make the timing of tasks an important concern in the case of SNs. In DQP this is not an issue, as these decisions are delegated to the OS/network layers.

```
BUILD-AGENDA($D_Q$, $R_Q$, $\alpha$, $\beta$, Time)
      ▷ schedule leaf fragments first
 1    for $i \leftarrow 1$ to $\beta$
 2        do for each $s \in R_Q$.Sites
 3              do $nextSlot[s] \leftarrow \alpha * (i-1)$
 4           while    ▷ post-order traversing $D_Q$
                      ▷ let $f$ denote the current fragment
 5              do if $f$.IsLeaf = yes
 6                    then $s.f$.ActAt ← [ ]
 7                         for each $s \in f$.Sites
 8                             do $s.f$.ActAt.Append $nextSlot[s]$
 9                                $nextSlot[s] \leftarrow +$ Time($s.f$)
      ▷ schedule non-leaf fragments next
10    while   ▷ post-order traversing $R_Q$,
              ▷ let $s$ denote the current site
11        do while    ▷ post-order traversing $D_Q$
                      ▷ let $f$ denote the current fragment
12              do if $f \in s$.AssignedFragments
13                    then $f$.ActAt ← $nextSlot[s]$
14                         $nextSlot[s] \leftarrow +$ Time($f$)*$\beta$
              ▷ schedule comms between fragments
15            $s$.TX.ActAt ← max($nextSlot[s]$,$nextSlot[s$.Parent])
16            $s$.Parent.RX($s$).ActAt ← $s$.TX.ActAt
17            $nextSlot[s] \leftarrow +$ Time($s$.TX)
18            $nextSlot[s$.Parent]) ← + $s$.Parent.RX
19    return agenda
```

**Fig. 11.** The agenda construction algorithm

*Relationship to Related Work:* In TinyDB, cost models are used to determine an acquisition rate to meet a user-specified lifetime. The schedule of work for each site is then determined by its level in the routing tree and the acquisition rate, and tuples are transmitted downstream following every acquisition without any buffering. In contrast, our approach allows the optimizer to determine an appropriate level of buffering, given the delivery time constraints specified by the user, which results in significant energy savings as described in Section 4 without having to compromise the acquisition interval. Note that this differs from the orthogonal approach proposed in TiNA [26], which achieves energy savings by not sending a tuple if an attribute is within a given threshold with respect to the previous tuple. It would not be difficult to incorporate such a technique into the SNEEql optimizer for greater energy savings. Zadorozhny [16] addresses a subset of the when-scheduling problem; an algebraic approach to generating schedules with as many non-interfering, concurrent communications as possible, is proposed. It is functionally similar to the proposed BUILD-AGENDA algorithm, although it only considers the scheduling of communications, and not computations as we do.

## 3.3 Phase 3: Code Generation

Step 8 in Fig. 3 generates executable code for each site based on the distributed QEP, routing tree and agenda. The current implementation of SNEEql generates nesC [27] code for execution in TinyOS [28], a component-based, event-driven

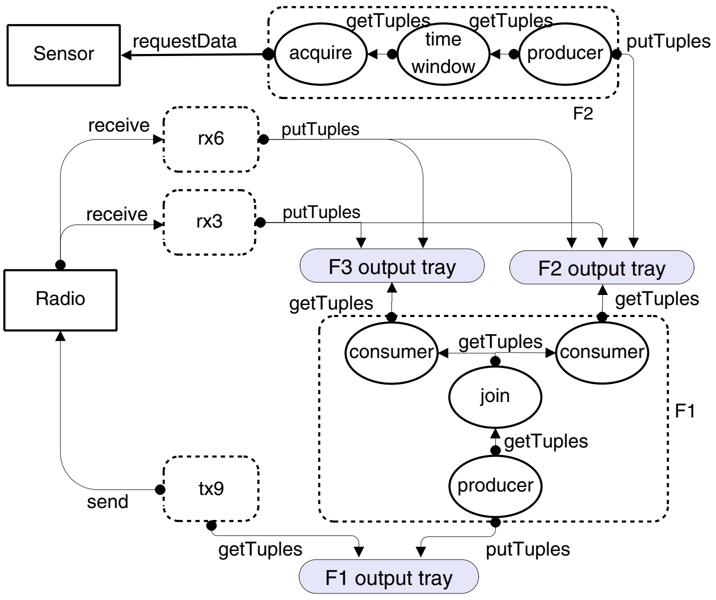


**Fig. 12.** Generated Code for Site 7 in Fig. 9

runtime environment designed for wireless SNs. nesC is a C-based language for writing programs over a library of TinyOS components (themselves written in nesC). Physical operators, such as those described in this and the previous section, are implemented as nesC template components. The code generator uses these component templates to translate the task-performing obligations in a site into nesC code that embodies the computing and communication activity depicted in abstract form by diagrams like the one in Fig. 12. The figure describes the activity in site 7, where the join (as well as sensing) is performed. In the figure, arrows denote component interaction, the black-circle end denoting the initiator of the interaction. The following kinds of components are represented in the figure: (i) square-cornered boxes denote software abstractions of hardware components, such as the sensor board and the radio; (ii) dashed, round-cornered boxes denote components that carry out agenda tasks in response to a clock event, such as a communication event or the evaluation of a QEP fragment; (iii) ovals denote operators which comprise fragments; note the the correspondence with Fig. 8 (recall that an EXCHANGE operator is typically implemented in two parts, referred to as producer and consumer, with the former communicating with the upstream fragment, and the latter, the downstream one); and (iv) shaded, round-cornered boxes denote (passive) buffers onto which tuples are written/pushed and from which tuples are read/pulled by other components.

Fig. 12 corresponds to the site 7 column in the agenda in Fig. 9 as follows. Firstly, the acquisitional fragment F2 executes twice and places the sensed tuples in the F2 output tray. Subsequently, tuples are received from sites 6 and 3, and are placed in the F2 output tray and F3 output tray accordingly. Inside fragment

F1, an exchange consumer gets tuples from F2 and another one gets tuples from F3 for the NL_JOIN. The results are fetched by a producer that writes them to the F1 output tray. Finally, tx9 transmits the tuples to site 9.

## 4 Experimental Evaluation

The goal of this section is to present experimental evidence we have collected in support of our overall research hypothesis, viz., that the extensions (described in Section 3) to DQP techniques that are proven in the case of robust networks lead to effective and efficient DQP over SNs. The experiments are analytical, i.e., aimed at collecting evidence as to the performance of the code produced by the SNEEql compiler/optimizer.

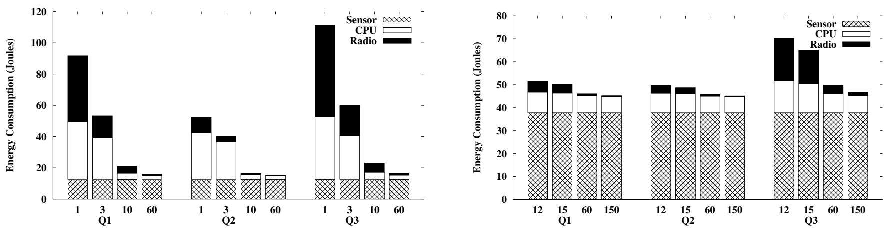
**Experimental Design.** Our experimental design centered around the aim of collecting evidence about the performance of our approach for a range of query constructs and across a range of QoS expectations. The QoS expectations we used were acquisition interval and delivery time. The evidence takes the form of measurements of the following performance indicators: network longevity, average delivery time, and total energy consumption. The queries used are generated from those in Fig. 1 (denoted Q1, Q2 and Q3 in the graphs) as follows: (1) in experiments where the acquisition rate is varied, the acquisition rate actually used in the query expression is not, of course, the one in Fig. 1, but should instead be read off the x-axis; and (2) varying the acquisition rate also requires that the startTime parameter in the TIME_WINDOW over the `inflow` extent of *Q3* is adjusted accordingly at each point in the x-axis, so that the scope and the acquisition rate are consistent with one another at every such point. In the experiments, we assume the selectivity of every predicate to be 1, i.e., every predicated evaluates to true for every tuple. This is so because it is the worst-case scenario in terms of the dependent variables we are measuring, viz., energy consumption and network longevity, as it causes the maximum amount of data to flow through the QEP (and hence through the routing tree formed for it over the underlying network).

**Experimental Set-Up.** The Experiments were run using Avrora [29], which simulates the behavior of SN programs at the machine code level with cycle-accuracy, and provides energy consumption data for each hardware component. The simulator ran executables compiled from TinyOS 1.1.15. All results are for 600 simulated seconds. A 10-node SN (depicted in Fig. 2 and the schema in Fig. 1) and 30-node SN are simulated in the experiments. The 30-node SN, not shown in this paper due to space limitations, has the same proportion of sources as the 10-node SN. The sensor nodes we have simulated were [Type = Mica2, CPU = 8-bit 7.3728MHz AVR, RAM = 4K, PM = 128K, Radio = CC1000, Energy Stock = 31320 J (2 Lithium AA batteries)].

**Experiment 1:** *Impact of acquisition interval on total energy consumption.* SNs often monitor remote or inaccessible areas (e.g., [19]), and a significant part of the total cost of ownership of a SN deployment is influenced by the energy stock required to keep it running. Results are reported in Fig. 13. The following can be observed: (i) As the acquisition rate $\alpha$ is increased, the total energy consumption decreases, as the query plan is acquiring, processing and transmitting less data overall, and sleeping for a longer proportion of the query evaluation process. (ii) A point is reached when increasing the acquisition rate leads to a marginally lower energy saving, due to the constant overhead incurred by having most of the components in a low-power state. The overhead is constant for the default Mica2 sensor board which is simulated, as it does not have a low power state, and is therefore always on. (iii) The radio energy consumption shrinks disproportionately compared to the CPU, because in relative terms, the energy saving when in a low power state is much greater for the radio than it is for the CPU. (iv) *Q2* has the lowest energy consumption because in the aggregation, the tuples from the source sites are reduced to a single tuple; in contrast, *Q3* joins tuples from the `inflow` and `outflow` extents which comprise all tuples in each acquisition, and therefore consumes the most energy.

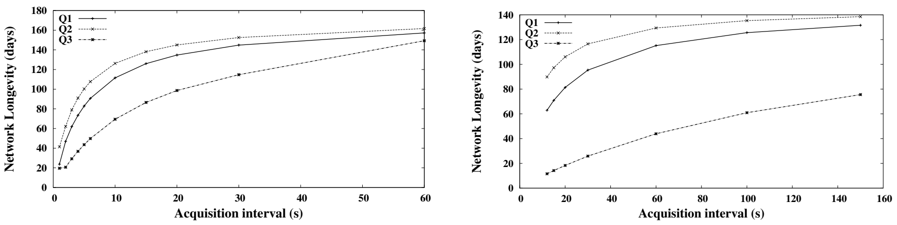
**Experiment 2:** *Impact of acquisition interval on network longevity.* The lifetime of a SN deployment is a vital metric as it indicates how often the SN energy stock will need to be replenished. Note that, here, network longevity is assumed to be the time it takes for the first node in the routing tree to fail, so this is, in a sense, a shortest-time-to-failure metric. Fig. 14 reports the results obtained. It can be observed that as acquisition interval increases, energy savings accrue, and hence network longevity increases, albeit with diminished returns, for the same reasons as in *Experiment 1.*

**Experiment 3:** *Impact of delivery time on network longevity.* For some applications, network longevity is of paramount importance and a delay in receiving the data may be tolerated (e.g., [19]), whereas in other applications it may be more important to receive the data quickly with less regard to preserving energy (e.g., [20]). Results which report the relationship between delivery time and longevity are shown in Fig. 15. It can be observed that: (i) The optimizer reacts to an increase in tolerable delivery time $\delta$ by increasing $\beta$, which in turn leads to an increase in network longevity – in other words, the optimizer empowers users, by enabling them to trade-off delivery time for greater network longevity; and (ii) Inflection points occur when increasing the buffering factor does not reduce energy consumption. This is because the when-scheduling algorithm assumes that increasing the buffering factor is always beneficial; this is however not always the case. A higher buffering factor may lead to a number of tuples being transmitted at a time that cannot be packed efficiently into a message, leading to padding in the message payload. As an example, consider the case where a single source node is being queried, $\beta = 4$, the number of tuples/message = 3. Packets will be formed of 3 tuples and then 1 tuple, which is inefficient. As a result, more messages need to be sent overall, leading to a higher energy consumption, and hence, a decreased network lifetime. This demonstrates that in order to ascertain an optimal buffering factor, minimizing the padding of messages is also an



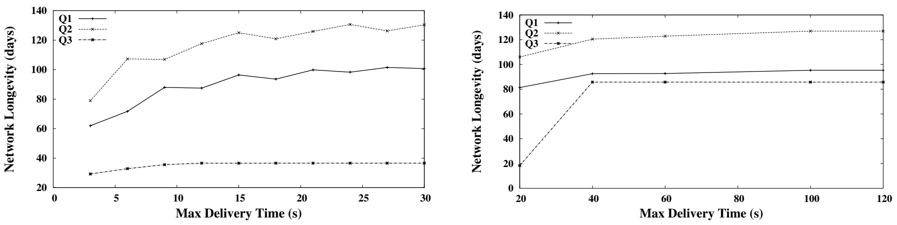
(a) 10 node network. (b) 30 node network.

**Fig. 13.** Energy consumption vs. $\alpha$ (in seconds)



(a) 10 node network. (b) 30 node network.

**Fig. 14.** Network Longevity vs. $\alpha$



(a) 10 node network. (b) 30 node network.

**Fig. 15.** Network Longevity vs. $\delta$

important consideration. However, we note that maximizing the buffering does lead to an overall improvement in network longevity.

To summarize, we can now ascertain that (i) the SNEEql optimizer exhibits desirable behaviors for a broad range of different scenarios; (ii) SNEEql allows different qualities of service (e.g., delivery time and lifetime) to be traded-off. This demonstrates that SNEEql delivers quantifiable benefits vis-à-vis the seminal contribution in the SN query processing area.

## 5 Conclusions

In this paper we have described SNEEql, a SN query optimizer based on the two-phase optimization architecture prevalent in DQP. In light of the differences between SNs and robust networks, we have highlighted the additional decision-making steps which are required, and the different criteria that need to be applied to inform decisions. We have demonstrated that, unlike TinyDB which performs very limited optimization, the staged decision-making approach in SNEEql offers benefits, including (1) the ability to schedule different workloads to different sites in the network, potentially enabling more economical use of resources such as memory, and to exploit heterogeneity in the SN, and (2) the ability to empower the user to trade-off conflicting qualities of service such as network longevity and delivery time. The effectiveness of the SNEEql approach of extending a DQP optimizer has been demonstrated through an empirical evaluation, in which the performance of query execution is observed to be well behaved under a range of circumstances. It can therefore be concluded that much can be learned from DQP optimizer architectures in the design of SN optimizer architectures.

**Acknowledgements.** This work is part of the SemSorGrid4Env project funded by the European Commission's Seventh Framework Programme, and the DIAS-MC project funded by the UK EPSRC WINES programme under Grant EP/C014774/1. We are grateful for this support and for the insight gained from discussions with our collaborators in the project. C.Y.A. Brenninkmeijer thanks the School of Computer Science, and F. Jabeen, the government of Pakistan, for their support.

## References

1. Karl, H., Willig, A.: Protocols and Architectures for Wireless Sensor Networks. John Wiley, Chichester (2005)
2. Ganesan, D., Mathur, G., et al.: Rethinking data management for storage-centric sensor networks. In: CIDR, pp. 22–31 (2007)
3. Madden, S., Franklin, M.J., et al.: Tinydb: An acquisitional query processing system for sensor networks. ACM Trans. Database Syst. 30(1), 122–173 (2005)
4. Yao, Y., Gehrke, J.: Query processing in sensor networks. In: CIDR, pp. 233–244 (2003)
5. Estrin, D., Culler, D., et al.: Connecting the physical world with pervasive networks. IEEE Pervasive Computing, 59–69 (January-March 2002)
6. Kossmann, D.: The state of the art in distributed query processing. ACM Comput. Surv. 32(4), 422–469 (2000)
7. Galpin, I., Brenninkmeijer, C.Y.A., et al.: An architecture for query optimization in sensor networks. In: ICDE, pp. 1439–1441 (2008)
8. Gehrke, J., Madden, S.: Query processing in sensor networks. In: IEEE Pervasive Computing, vol. 3, pp. 46–55. IEEE Computer Society, Los Alamitos (2004)
9. Brayner, A., Lopes, A., et al.: Toward adaptive query processing in wireless sensor networks. Signal Processing 87(12), 2911–2933 (2007)
10. Deshpande, A., Guestrin, C., et al.: Model-based approximate querying in sensor networks. VLDB J. 14(4), 417–443 (2005)
11. Trigoni, N., Yao, Y., et al.: Wavescheduling: Energy-efficient data dissemination for sensor networks. In: DMSN, pp. 48–57. ACM Press, New York (2004)
12. Madden, S., Franklin, M.J.: et al.: Tag: A tiny aggregation service for ad-hoc sensor networks. In: OSDI, pp. 131–146 (2002)

13. Manjhi, A., Nath, S.: et al.: Tributaries and deltas: Efficient and robust aggregation in sensor network streams. In: SIGMOD, pp. 287–298 (2005)
14. Trigoni, N., Yao, Y., et al.: Multi-query optimization for sensor networks. In: DCOSS, pp. 307–321 (2005)
15. Bonfils, B.J., Bonnet, P.: Adaptive and decentralized operator placement for in-network query processing. In: Zhao, F., Guibas, L.J. (eds.) IPSN 2003. LNCS, vol. 2634, pp. 47–62. Springer, Heidelberg (2003)
16. Zadorozhny, V.I., Chrysanthis, P.K., et al.: A framework for extending the synergy between query optimization and mac layer in sensor networks. In: DMSN, pp. 68–77. ACM Press, New York (2004)
17. Arasu, A., Babcock, B., et al.: Stream: The stanford stream data manager. IEEE Data Eng. Bull. 26(1), 19–26 (2003)
18. Brenninkmeijer, C.Y.A., Galpin, I., et al.: A semantics for a query language over sensors, streams and relations. In: Gray, A., Jeffery, K., Shao, J. (eds.) BNCOD 2008. LNCS, vol. 5071, pp. 87–99. Springer, Heidelberg (2008)
19. Marshall, I.W., Price, M.C., et al.: Multi-sensor cross correlation for alarm generation in a deployed sensor network. In: Kortuem, G., Finney, J., Lea, R., Sundramoorthy, V. (eds.) EuroSSC 2007. LNCS, vol. 4793, pp. 286–299. Springer, Heidelberg (2007)
20. Stoianov, I., Nachman, L., et al.: Pipenet: a wireless sensor network for pipeline monitoring. In: IPSN, pp. 264–273 (2007)
21. Sampaio, S.F.M., Paton, N.W., et al.: Measuring and modelling the performance of a parallel odmg compliant object database server. CCPE 18(1), 63–109 (2006)
22. Stonebraker, M., Aoki, P.M., et al.: Mariposa: A wide-area distributed database system. VLDB J. 5(1), 48–63 (1996)
23. Graefe, G.: Encapsulation of parallelism in the volcano query processing system. In: SIGMOD, pp. 102–111 (1990)
24. Gounaris, A., Sakellariou, R., et al.: A novel approach to resource scheduling for parallel query processing on computational grids. DPD 19(2-3), 87–106 (2006)
25. Smith, J., Gounaris, A., Watson, P., et al.: Distributed query processing on the grid. In: GRID, pp. 279–290 (2002)
26. Sharaf, M.A., Beaver, J., et al.: Tina: A scheme for temporal coherency-aware in-network aggregation. In: MobiDE, pp. 69–76 (2003)
27. Gay, D., Levis, P., et al.: The nesc language: A holistic approach to networked embedded systems. In: PLDI, pp. 1–11 (2003)
28. Hill, J., Szewczyk, R., et al.: System architecture directions for networked sensors. In: ASPLOS, pp. 93–104 (2000)
29. Titzer, B., Lee, D.K., et al.: Avrora: scalable sensor network simulation with precise timing. In: IPSN, pp. 477–482 (2005)

# Efficient Evaluation of Generalized Tree-Pattern Queries with Same-Path Constraints

Xiaoying Wu[1], Dimitri Theodoratos[1], Stefanos Souldatos[2], Theodore Dalamagas[3], and Timos Sellis[2,3]

[1] New Jersey Institute of Technology, USA
xw43@njit.edu, dth@cs.njit.edu
[2] National Technical University of Athens, Greece
stef@dblab.ece.ntua.gr
[3] Institute for the Management of Information Systems, Greece
dalamag@imis.athena-innovation.gr,
timos@imis.athena-innovation.gr

**Abstract.** Querying XML data is based on the specification of structural patterns which in practice are formulated using XPath. Usually, these structural patterns are in the form of trees (Tree-Pattern Queries – TPQs). Requirements for flexible querying of XML data including XML data from scientific applications have motivated recently the introduction of query languages that are more general and flexible than TPQs. These query languages correspond to a fragment of XPath larger than TPQs for which efficient non-main-memory evaluation algorithms are not known.

In this paper, we consider a query language, called Partial Tree-Pattern Query (PTPQ) language, which generalizes and strictly contains TPQs. PTPQs represent a broad fragment of XPath which is very useful in practice. We show how PTPQs can be represented as directed acyclic graphs augmented with "same-path" constraints. We develop an original polynomial time holistic algorithm for PTPQs under the inverted list evaluation model. To the best of our knowledge, this is the first algorithm to support the evaluation of such a broad structural fragment of XPath. We provide a theoretical analysis of our algorithm and identify cases where it is asymptotically optimal. In order to assess its performance, we design two other techniques that evaluate PTPQs by exploiting the state-of-the-art existing algorithms for smaller classes of queries. An extensive experimental evaluation shows that our holistic algorithm outperforms the other ones.

## 1 Introduction

XML data are often encountered in e-science (astronomy, biology, earth science, etc.), natural language processing, digital entertainment, social network analysis, and more. Querying XML data is based on the specification of structural patterns. In practice, these structural patterns are specified using XPath [1], a language that lies at the core of the standard XML query language XQuery [1]. Usually, the structural patterns are in the form of trees (Tree-Pattern Queries – TPQs). A restrictive characteristic of TPQs is that they impose a total order for the nodes in every path of the query pattern. However, recent applications of XML require querying of data whose structure is complex [2] or is not fully known to the user [3,4,5,6], or integrating XML data sources with different structures [7,3,4]. In order to satisfy these requirements, different approaches are adopted that range from using unstructured keyword queries [7] to extending XQuery with keyword search capabilities [8,3]. TPQs are not expressive enough to specify these new types of queries. Larger subclasses of XPath are required for which, up to now, efficient non-main-memory evaluation algorithms are not known.

In this paper, we consider a query language for XML, called Partial Tree-Pattern Query (PTPQ) language. PTPQs generalize and strictly contain TPQs. They are flexible enough to allow a large range of queries from keyword-style queries with no structure, to keyword queries with arbitrary structural constraints, to fully specified TPQs. PTPQs are not restricted by a total order for the nodes in a path of the query pattern since they can constrain a number of (possibly unrelated) nodes to lie on the same path (*same-path* constraint). These nodes together form a *partial path*. PTPQs can express XPath queries with the reverse axes parent and ancestor, in addition to forward child and descendant axes and branching predicates. They can also express the node identity equality (*is-same-node*) operator of XPath by employing *node sharing expressions*. Overall, PTPQs represent a broad fragment of XPath which is very useful in practice.

A broad fragment of XPath such as PTPQs can be useful only if it is complemented with efficient evaluation techniques. A growing number of XML applications, in particular data-centric applications, handle documents too large to be processed in memory. This is usually the case with scientific applications.

A recent approach for the non-main-memory evaluation of queries on XML data assumes that the data is preprocessed and the position of every node in the XML document tree is encoded [9,10,11]. Further, an inverted list is built on every node label. In order to evaluate a query, the nodes of the relevant inverted lists are read in the preorder of their appearance in the XML tree. We refer to this evaluation model as *inverted lists* model. Algorithms in this model [9,10,11] are based on stacks that allow encoding an exponential number of pattern matches in a polynomial space. The advantage of the inverted lists evaluation is that it can process large XML documents without preloading them in the memory (non-main-memory evaluation). Unfortunately, existing non-main-memory evaluation algorithms focus almost exclusively on TPQs.

**Problem addressed.** In this paper we undertake the task of designing an efficient evaluation algorithm for PTPQs in the inverted lists model. This task is complex: as we show later, due to their expressive power, PTPQs can only be represented as directed acyclic graphs (dags) annotated with same-path constraints. Matching these query patterns to XML trees requires the appropriate handling of both the structural constraints of the dag, and the same-path constraints. These two types of constraints can be conflicting: a matching that satisfies the structural constraints of the dag may violate the same-path constraints, and vice versa.

One might wonder whether existing techniques can be used for efficiently evaluating PTPQs. In fact, as we show later in the paper, a PTPQ is equivalent to a set of TPQs for which efficient algorithms exist. Unfortunately, this transformation leads to a number of TPQs which, in the worst case, is exponential on the size of the PTPQ. Our experimental results show that another technique that decomposes the PTPQ dag into simpler query patterns, which can be evaluated efficiently, also fails to produce satisfactory performance.

**Contribution.** The main contributions of the paper are:

- We use a formalism to represent PTPQs as directed acyclic graphs (dags) annotated with same-path constraints (Section 3.2). We show that PTPQs can express a broad fragment of XPath which comprises reverse axes and the node identity equality (*is-same-node*) operator in addition to forward axes and predicates (Section 3.3).
- We develop an efficient holistic evaluation algorithm for PTPQs called $PartialTreeStack$ (Section 5.1). $PartialTreeStack$ takes into account the dag form of PTPQs and avoids redundant processing of subdags having multiple "parents". It wisely avoids checking whether node matches satisfy the dag structural constraints when it can derive that they violate a same-path constraint. $PartialTreeStack$ finds solutions for the partial paths of the query and merge-joins them to produce the query answer. When no parent-child relationships are present in the query dag, it is guaranteed that every partial path solution produced will participate in the final answer. Therefore, $PartialTreeStack$ does not produce intermediate results.
- We provide a theoretical analysis of $PartialTreeStack$ to show its polynomial time and space complexity. We further show that under the reasonable assumption that the size of queries is not significant compared to the size of data, $PartialTreeStack$ is asymptotically optimal for PTPQs without parent-child structural relationships (Section 5.3)
- In order to assess the performance of $PartialTreeStack$, we design, for comparison, two approaches that exploit existing state-of-the-art techniques for more restricted classes of queries (Section 6.1): algorithm $TPQGen$, generates a set of TPQs equivalent to the given PTPQ, and computes the answer of the PTPQ by taking the union of their solutions. Algorithm $PartialPathJoin$ decomposes the PTPQ into partial-path queries and computes the answer of the PTPQ by merge-joining their solutions.
- We implemented all three algorithms and conducted detailed experiments to compare their performance. The experimental results show that $PartialTreeStack$ outperforms the other two algorithms (Section 6.2).
- To the best of our knowledge, $PartialTreeStack$ is the first algorithm in the inverted lists model that supports such a broad fragment of XPath.

## 2 Related Work

In this paper, we assume that queries are evaluated in the inverted lists evaluation model. This evaluation model uses inverted lists built over the input data to avoid: (1) preloading XML documents in memory, and (2) processing large portions of the XML documents that are not relevant to the query evaluation. Because of these advantages, many query evaluation algorithms for XML have been developed in this model. These algorithms broadly fall in two categories: the *structural join* approach [9,12], and the *holistic twig join* approach [10,13,14,15,16,14]. All these algorithms, however, focus almost exclusively on TPQs.

The *structural join* approach first decomposes a TPQ into a set of binary descendant or child relationships. Then, it evaluates the relationships using binary merge join. This approach might not be efficient because it generates a large number of intermediate solutions (that is, solutions for the binary relationships that do not make it to the answer of the TPQ). Algorithms for structural join order optimization were introduced in [12].

The *holistic twig join* approach (e.g. $TwigStack$ [10]) represents the state of the art for evaluating TPQs. This approach evalutes TPQs by joining multiple input lists at a time to avoid producing large intermediate solutions. Algorithm $TwigStack$ is shown optimal for TPQs without child relationships.

Several papers focused on extending *TwigStack*. For example, in [13], algorithm *TwigStackList* evaluates efficiently TPQs in the presence of child relationships. Algorithm $iTwigJoin$ extended $TwigStack$ by utilizing structural indexes built on the input lists [14]. Evaluation methods of TPQs with OR predicates were developed in [15].

All the above algorithms are developed for TPQs and cannot be used nor extended so that they evaluate PTPQs. The reason is that PTPQs are not mere tree patterns but dags augmented with same-path constraints. Chen et al. [16] proposed twig join algorithms that handle dag queries over graph structured data. Note that, the semantics of the dag queries dealt with in [16] is different than the semantics of PTPQ dag queries studied in this paper since their dag queries are matched against XML graphs and not trees.

Considerable work has also been done on the processing of XPath queries when the XML data is not encoded and indexed (main-memory evaluation or streaming evaluation). For example, [17] suggested polynomial main-memory algorithms for answering full XPath queries. The streaming evaluation, though a single choice for a number of applications, cannot be compared in terms of performance to the inverted lists evaluation we adopt here. The reason is that in the streaming evaluation, no indexes or inverted lists can be exploited and the whole XML document has to be sequentially scanned.

PTPQs were initially introduced in [4]. Their containment problem was studied in [18] and PTPQ semantic issues were addressed in [5]. Relevant to our work are also the evaluation algorithms for partial path queries [19,20]. Partial path queries are not a subclass of TPQs but they form a subclass of PTPQs.

## 3 Data Model and Partial Tree Pattern Query Language

### 3.1 Data Model

XML data is commonly modeled by a tree structure. Tree nodes are labeled and represent elements, attributes, or values. Let $\mathcal{L}$ be the set of node labels. Tree edges represent element-subelement, element-attribute, and element-value relationships. Without loss of generality, we assume that only the root node of every XML tree is labeled by $r \in \mathcal{L}$. We denote XML tree labels by lower case letters. To distinguish between nodes with the same label, every node in the XML tree has an identifier shown as a subscript of the node label. For XML trees, we adopt the positional representation widely used for XML query processing [9,10,11]. The positional representation associates with every node a triplet (*start*,*end*,*level*) of values. The *start* and *end* values of a node are integers which can be determined through a depth-first traversal of the XML tree, by sequentially assigning numbers to the first and the last visit of the node. The *level* value represents the level of the node in the XML tree.

The positional representation allows efficiently checking structural relationships between two nodes in the XML tree. For instance, given two nodes $n_1$ and $n_2$, $n_1$ is an ancestor of $n_2$ iff $n_1.start < n_2.start$, and $n_2.end < n_1.end$. Node $n_1$ is the parent of $n_2$ iff $n_1.start < n_2.start$, $n_2.end < n_1.end$, and $n_1.level = n_2.level - 1$.

In this paper, we often need to check whether a number of nodes in an XML tree lie on the same path. This check can be performed efficiently using the following proposition.

**Proposition 1.** *Given a set of nodes* $n_1, \ldots, n_k$ *in an XML tree* $T$*, let* maxStart *and* minEnd *denote respectively the maximum start and the minimum end values in the positional representations of* $n_1, \ldots, n_k$*. Nodes* $n_1, \ldots, n_k$ *lie on the same path in* $T$ *iff* maxStart $\leq$ minEnd.

## 3.2 Query Language

**Syntax.** A partial tree-pattern query (PTPQ) specifies a pattern which partially determines a tree. PTPQs comprise nodes and child and descendant relationships between nodes. The nodes are grouped into disjoint sets called *partial paths*. PTPQs are embedded to XML trees. The nodes of a partial path are embedded to nodes on the *same* XML tree path. However, unlike paths in TPQs the child and descendant relationships in partial paths do not necessarily form a total order. This is the reason for qualifying these paths as partial. PTPQs also comprise node sharing expressions. A node sharing expression indicates that two nodes from different partial paths are to be embedded to the same XML tree node. That is, the image of these two nodes is the same – *shared* – node in the XML tree. The formal definition of a PTPQ follows.

**Definition 1 (PTPQ).** *Let* $\mathcal{N}$ *be an infinite set of labeled nodes. Nodes in* $\mathcal{N}$ *are labeled by a label in* $\mathcal{L}$*. Let* $X$ *and* $Y$ *denote distinct nodes in* $\mathcal{N}$*. A* partial tree-pattern query *is a pair* $(\mathbf{S}, N)$ *where:*

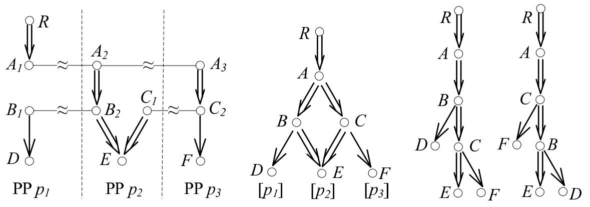
**S** *is a list of* $n$ *named sets* $p_1, \ldots, p_n$ *called* partial paths *(*PPs*). Each PP* $p_i$ *is a finite set of expressions of the form* $X/Y$ *(*child relationship*) or* $X//Y$ *(*descendant relationship*). We write* $X[p_i]/Y[p_i]$ *(resp.* $X[p_i]//Y[p_i]$*) to indicate that* $X[p_i]/Y[p_i]$ *(resp.* $X[p_i]//Y[p_i]$*) is a relationship in PP* $p_i$*. Child and descendant relationships are collectively called* structural *relationships.*

$N$ *is a set of* node sharing *expressions* $X[p_i] \approx Y[p_j]$*, where* $p_i$ *and* $p_j$ *are distinct PPs, and* $X$ *and* $Y$ *are nodes in PPs* $p_i$ *and* $p_j$ *respectively such that both of them are labeled by the same label in* $\mathcal{L}$*.*

Figure 1(a) shows a PTPQ $Q_1$ and Figure 1(b) shows the visual representation of $Q_1$. We use this representation later on in Section 6 to design a comparison algorithm for evaluating PTPQs. Unless otherwise indicated, in the following, “query” refers to a PTPQ. Note that the labels of the query nodes are denoted by capital letters to distinguish them from the labels of the XML tree nodes. In this sense, label $l$ in an XML tree and label $L$ in a query represent the same label.

$$
\begin{aligned}
\mathbf{S} = \{ \\
&p_1\text{: } \{ R[p_1]//A_1[p_1],\ B_1[p_1]/D[p_1] \}, \\
&p_2\text{: } \{ A_2[p_2]//B_2[p_2],\ B_2[p_2]//E[p_2], \\
&\qquad C_1[p_2]//E[p_2] \}, \\
&p_3\text{: } \{ A_3[p_3]//C_2[p_3],\ C_2[p_3]/F[p_3] \} \\
&\} \\
N = \{ &B_1[p_1] \approx B_2[p_2],\ C_1[p_2] \approx C_2[p_3] \\
&A_1[p_1] \approx A_2[p_2],\ A_2[p_2] \approx A_3[p_3] \}
\end{aligned}
$$

(a) PTPQ $Q_1$



(b) Visual representation of $Q_1$ (c) Query graph of $Q_1$ (d) The two TPQs of $Q_1$

**Fig. 1.** A PTPQ and its three representations that are used by different algorithms in the paper

**Semantics.** The answer of a PTPQ on an XML tree is a set of tuples of nodes from the XML tree that satisfy the structural relationships and the same path constraints of the PTPQ. Formally:

**Definition 2 (Query Embedding).** *An* embedding *of a query $Q$ into an XML tree $T$ is a mapping $M$ from the nodes of $Q$ to nodes of $T$ such that: (a) a node $A[p_j]$ in $Q$ is mapped by $M$ to a node of $T$ labeled by a; (b) the nodes of $Q$ in the same PP are mapped by $M$ to nodes that lie on the same path in $T$; (c) $\forall\ X[p_i]/Y[p_i]$ (resp. $X[p_i]//Y[p_i]$) in $Q$, $M(Y[p_i])$ is a child (resp. descendant) of $M(X[p_i])$ in $T$; (d) $\forall\ X[p_i] \approx Y[p_j]$ in $Q$, $M(X[p_i])$ and $M(Y[p_j])$ coincide in $T$.*

We call *image* of $Q$ under an embedding $M$ a tuple that contains one field per node in $Q$, and the value of the field is the image of the node under $M$. Such a tuple is also called *solution* of $Q$ on $T$. The *answer* of $Q$ on $T$ is the set of solutions of $Q$ under all possible embeddings of $Q$ to $T$.

**Graph representation for PTPQs.** For our evaluation algorithm, we represent queries as node labeled annotated directed graphs: a query $Q$ is represented by a graph $Q_G$. Every node $X$ in $Q$ corresponds to a node $X_G$ in $Q_G$, and vice versa. Node $X_G$ is labeled by the label of $X$. Two nodes in $Q$ participating in a node sharing expression correspond to the same node in $Q_G$. Otherwise, they correspond to distinct nodes in $Q_G$. For every structural relationship $X//Y$ (resp. $X/Y$) in $Q$ there is a single (resp. double) edge in $Q_G$. In addition, each node in $Q_G$ is annotated by the set of PPs of the nodes in $Q$ it corresponds to. Note that these annotations allow us to express same-path constraints. That is, all the nodes annotated by the same partial path have to be embedded to nodes in an XML tree that lie on the same path.

Figure 1(c) shows the query graph of query $Q_1$ of Figure 1(a). Note that a node in the graph inherits all the annotating PPs of its descendant nodes. Because of this inheritance property of partial path annotations we can omit in the figures the annotation of internal nodes in queries when no ambiguity arises. For example, in the graph of Figure 1(c), node $A$ is annotated by the PPs $p_1$, $p_2$, and $p_3$ inherited from its descendant nodes $D$, $E$, and $F$.

Clearly, a query that has a cycle is unsatisfiable (i.e., its answer is empty on any XML tree). Therefore, in the following, we assume a query is a dag and we identify a query with its dag representation.

### 3.3 Generality of Partial Tree Pattern Query Language

Clearly, the class of PTPQs cannot be expressed by TPQs. For instance, PTPQs can constrain a number of nodes in a query pattern to belong to the same path even if there is no precedence relationship between these nodes in the PTPQ. Such a query cannot be expressed by a TPQ. TPQs correspond to the fragment XP$^{\{[],/,//\}}$ of XPath that involves predicates([]), and child (/) and descendant (//) axes. In fact, it is not difficult to see that PTPQs cannot be expressed either by the larger fragment XP$^{\{[],/,//,\backslash,\backslash\backslash\}}$ of XPath that involves, in addition, the reverse axes parent (\) and ancestor (\\). On the other hand, PTPQs represent a very broad fragment XP$^{\{[],/,//,\backslash,\backslash\backslash,\approx\}}$ of XPath that corresponds to XP$^{\{[],/,//,\backslash,\backslash\backslash\}}$ augmented with the $is$ operation ($\approx$) of XPath2 [1]. The $is$ operator is a node identity equality operator. The conversion of an expression in XP$^{\{[],/,//,\backslash,\backslash\backslash,\approx\}}$ to an equivalent PTPQ is straightforward. There is no previous inverted lists evaluation algorithm that directly supports such a broad fragment of XPath.

Note that as the next proposition shows, a PTPQ is equivalent to a *set* of TPQs.

**Proposition 2.** *Given a PTPQ $Q$ there is a set of TPQs $Q_1, \dots, Q_n$ in XP$^{\{[],/,//\}}$ such that for every XML tree $T$, the answer of $Q$ on $T$ is the union of the answers of the $Q_i$s on $T$.*

As an example, Figure 1(d) shows the two TPQs for query $Q_1$ of Figure 1(a), which together are equivalent to $Q_1$. Based on the previous proposition, one can consider evaluating PTPQs using existing algorithms for TPQs. In Section 6.1, we present such an algorithm. However, the number of TPQs that need to be evaluated can grow to be large (in the worst case, it can be *exponential* on the number of nodes of the PTPQ). Therefore, the performance of such an algorithm is not expected to be satisfactory. In Section 5, we present our novel holistic algorithm, *PartialTreeStack*, that efficiently evaluates PTPQs in the inverted lists evaluation model.

## 4 Data Structures and Functions for PTPQ Evaluation

We present in this section the data structures and operations we use for PTPQ evaluation in the inverted lists model.

**Query functions.** Let $Q$ be a query, $X$ be a node in $Q$, and $p_i$ be a partial path in $Q$. Node $X$ is called *sink node of* $p_i$, if $p_i$ annotates $X$ but no any descendant nodes of $X$ in $Q$. We make use of the following functions in the evaluation algorithm.

Function *sinkNodes*($p_i$) returns the set of sink nodes of $p_i$. Function *partialPaths*($X$) returns the set of partial paths that annotate $X$ in $Q$ and *PPsSink*($X$) returns the set of partial paths where $X$ is a sink node. Boolean function $isSink(X)$ returns $true$ iff $X$ is a sink node in $Q$ (i.e., it does not have outgoing edges in $Q$). Function *parents*($X$) returns the set of parent nodes of $X$ in $Q$. Function *children*($X$) returns the set of child nodes of $X$ in $Q$.

**Operations on inverted lists.** With every query node $X$ in $Q$, we associate an inverted list $T_X$ of the positional representation of the nodes labeled by $x$ in the XML tree. The nodes in $T_X$ are ordered by the their *start* field (see Section 3). To access sequentially the nodes in $T_X$, we maintain a cursor. We use $C_X$ to denote the node currently pointed by the cursor in $T_X$ and call it the *current match* of $X$. Operation *advance*($X$) moves the cursor to the next node in $T_X$. Function *eos*($X$) returns true if the cursor has reached the end of $T_X$.

**Stacks.** With every query node $X$ in $Q$, we associate a stack $S_X$. An entry $e$ in stack $S_X$ corresponds to a node in $T_X$ and has the following two fields:
1. A field consisting of the triplet $(start,\ end,\ level)$ which is the positional representation of the corresponding node in $T_X$.
2. A field $ptrs$ which is an array of pointers indexed by $parents(X)$. Given $P \in parents(X)$, $ptrs[P]$ points to the highest among the entries in stack $S_P$ that correspond to ancestors of $e$ in the XML tree.

**Stack operations.** We use the following stack operations: *push*($S_X$,$entry$) which pushes $entry$ on the stack $S_X$, *top*($S_X$) which returns the top entry of stack $S_X$, and *bottom*($S_X$) which returns the bottom entry of stack $S_X$. Boolean function *empty*($S_X$) returns $true$ iff $S_X$ is empty.

Initially, all stacks are empty, and for every query node $X$, its cursor points to the first node in $T_X$. At any point during the execution of the algorithm, the entries that stack $S_X$ can contain correspond to nodes in $T_X$ before the current match $C_X$. The entries in a stack below an entry $e$ are ancestors of $e$ in the XML tree. Stack entries form partial solutions of the query that can be extended to become the solutions as the algorithm goes on.

**Matching query subdags.** Recall that $C_X$ denotes the current match of the query node $X$. Below, we define a concept which is important for understanding the query evaluation algorithm.

**Definition 3 (Current Binding).** *Given a query Q, let $X$ be a node in Q and $Q_X$ be the subdag (subquery) of Q rooted at $X$. The* current binding of $Q$ *is the tuple $\beta$ of current matches of the nodes in Q. Node $X$ is said to* have a solution in $\beta$, *if the matches of the nodes of $Q_X$ in $\beta$ form a solution for $Q_X$.*

If node $X$ has a solution in $\beta$, then the following two properties hold: (1) $C_X$ is the ancestor of all the other current matches of the nodes in $Q_X$, and (2) current matches of the query nodes in $Q_X$ in the same partial path lie on the same path in the XML tree.

When all the structural relationships in $Q$ are regarded as descendant relationships, we can show the following proposition.

**Proposition 3.** *Let $X$ be a node in a query Q where all the structural relationships are regarded as descendant relationships, $\{Y_1, \dots, Y_k\}$ be the set of child nodes of $X$ in Q, and $\{p_1, \dots, p_n\}$ be the set of partial paths annotating $X$ in Q. Let also $\beta$ denote the current binding of Q. Node $X$ has a solution in $\beta$ if and only if the following three conditions are met:*
1. *All $Y_i$s have a solution in $\beta$.*
2. *$C_X$ is a common ancestor of all $C_{Y_i}$s in the XML tree.*
3. *For each partial path $p_j$, the current matches of all the sink nodes of $p_j$ that are descendants of $X$ lie on the same path in the XML tree.*

The proof follows directly from Definition 3. Clearly, if $X$ is a sink node, it satisfies the conditions of Proposition 3, and therefore, it has a solution in $\beta$.

As an example for Proposition 3, consider evaluating query $Q_3$ of Figure 4(b) on the XML tree of Figure 4(a). Suppose the cursors of $R$, $A$, $B$, $D$, $C$, $E$, $G$, and $F$ are at $r$, $a_1$, $b_1$, $d_1$, $c_1$, $e_1$, $g_1$, and $f_1$, respectively. By Proposition 3, node $D$ has a solution in the the current binding $\beta$ of $Q_3$, since (1) child nodes $E$ and $F$ both have a solution in $\beta$; (2) $b_1$ is a common ancestor of $e_1$ and $f_1$; and (3) $E$ and $F$ are the only descendant sink nodes of $D$ in partial paths $p_1$ and $p_2$, respectively. However, node $B$ does not have a solution in $\beta$ because the condition 3 of Proposition 3 is violated: $g_1$ and $f_1$, which respectively are the current matches of the descendant sink nodes $G$ and $F$ in partial path $p_2$, are not on the same path in the XML tree.

## 5 PTPQ Evaluation Algorithm

The flexibility of the PTPQ language in specifying queries and its increased expressive power makes the design of an evaluation algorithm challenging. Two outstanding reasons of additional difficulty are: (1) a query is a dag (which in the general case is not merely a tree) augmented with constraints, and (2) the same-path constraints should be enforced for all the nodes in a partial path in addition to enforcing structural relationships. In this section, we present our holistic evaluation algorithm $PartialTreeStack$, which efficiently resolves these issues. The presentation of the algorithm is followed by an analysis of its correctness and complexity.

### 5.1 Algorithm PartialTreeStack

Algorithm $PartialTreeStack$ operates in two phases. In the first phase, it iteratively calls a function called $getNext$ to identify the next query node to be processed. Solutions to individual partial paths of the query are also computed in this phase. In the second phase, the partial path solutions are merge-joined to compute the answer of the query.

**Function *getNext*.** Function $getNext$ is shown in Listing 1. It is called on a query node and returns a query node (or $null$). Starting with the root $R$ of the query dag $Q$, function $getNext$ traverses the dag in left-right and depth-first search mode. For every node under consideration, $getNext$ recursively calls itself on each child of that node. This way, $getNext$ first reaches the left-most sink node of $Q$. Starting from that sink node, it tries to find a query node $X$ with the following three properties:

1. $X$ has a solution in the current binding $\beta$ of $Q$ but none of $X$'s parents has a solution in $\beta$.
2. Let $P$ be a parent of $X$ in the invocation path of $getNext$. The current match of $X$, i.e., $C_X$, has the smallest $start$ value among the current matches of all the child nodes of $P$ that have a solution in $\beta$.
3. For each partial path $p_i$ annotating $X$, $C_X$ has the smallest $start$ value among the current matches of all the nodes annotated by $p_i$ that have a solution in $\beta$.

**Listing 1.** Function getNext($X$)

```
if (isSink($X$) $\vee$ $knownSoln[X]$ ) then
    return $X$
for ($Y_i \in$ children($X$)) do
    $invPath[Y_i] \leftarrow invPath[X]$ + '$Y_i$'
    $Y \leftarrow$ getNext($Y_i$)
    if ($Y \neq Y_i \wedge Y \neq X$ ) then
        return $Y$
$Y_{min} \leftarrow$ minarg$_{Y_i}\{C_{Y_i}.start\}$, $Y_{max} \leftarrow$ maxarg$_{Y_i}\{C_{Y_i}.start\}$, where $Y_i \in$ children($X$) $\wedge$ $knownSoln[Y_i]$
while ($C_X.end < C_{Y_{max}}.start$) do
    advance($X$)
if ($C_X.start < C_{Y_{min}}.start$) then
    updateSPStatus($X$)
    if ($\forall p_i \in$ partialPaths($X$): $SP[p_i]$) then
        $knownSoln[X] \leftarrow true$
        return $X$
    else
        return $null$
if (bottom($S_X$) is an ancestor of $C_{Y_{min}}$) then
    if ($\exists P \in$ parents($Y_{min}$): $C_P$ is an ancestor of $C_{Y_{min}}$) then
        return the lowest ancestor of $P$ among the nodes in $invPath[X]$
    if ($\exists$ sink node Z $\in Q$: partialPaths($Z$)$\subseteq$ partialPaths($Y_{min}$)$\wedge$ $C_Z.start < C_{Y_{min}}.start$) then
        return the lowest ancestor of $Z$ among the nodes in $invPath[X]$
if ($\forall p_i \in$ partialPaths($Y_{min}$): $SP_{Y_{min}}[X, p_i] \neq null$) then
    return $Y_{min}$
updateSPStatus($X$)
if ($\forall p_i \in$ partialPaths($X$): $SP[p_i]$ )) then
    return $Y_{min}$
else
    return $null$
```

Node $X$ is the node returned by $getNext(R)$ to the main algorithm for processing. The first property guarantees that: (1) $C_X$ is in a solution of $Q_X$, and (2) a query node match in a solution of $Q$ is always returned before other query node matches in the same solution that are descendants of it in the XML tree. The third property guarantees that matches of query nodes annotated by the same partial path are returned in the order of their $start$ value (i.e., according to the pre-order traversal of the XML tree).

During the traversal of the dag, function $getNext$ discards node matches that are guaranteed not to be part of any solution of the query by advancing the corresponding cursors. This happens when a structural constraint of the dag or a same-path constraint is violated.

**Dealing with the query dag.** Since $Q$ is a dag, some nodes of $Q$ along with their subdags could be visited multiple times by $getNext$ during its traversal of $Q$. This happens when a node has multiple parents in $Q$. Figure 2 shows a scenario of the traversal of a query dag by $getNext$, where node $X$ has parents $P_1, \ldots, P_k$. Function $getNext$ will be called on $X$ from each one of the $k$ parents of $X$. To prevent redundant computations, a boolean array, called $knownSoln$, is used. Array $knownSoln$ is indexed by the nodes of $Q$. Given a node $X$ of $Q$, if $knownSoln[X]$ is $true$, $getNext$ has already processed the subdag $Q_X$ rooted at $X$, and $X$ has a solution in the current binding $\beta$ of $Q$. In this case, subsequent calls of $getNext$ on $X$ from other parents of $X$ are not processed on the subdag $Q_X$ since they are known to return $X$ itself.

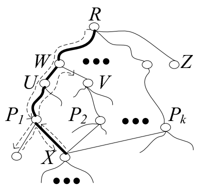


**Fig. 2.** Traversal of a query dag by $getNext$

The traversal of the query nodes is not necessarily in accordance with the pre-order traversal of the query node matches in the XML tree. It is likely that the current match of a node $X$ already visited by $getNext$ has larger $start$ value than that of a node that has not been visited yet. If this latter node is an ancestor of $X$ and has a match that

**Listing 2.** Procedure updateSPStatus($X$)

```
1  for ($p_i \in$ partialPaths($X$)) do
2      let $nodes$ denote the set of sink nodes of $p_i$ that are descendants of $X$ in $Q$
3      let $node$ denote the node in $nodes$ whose current match has the smallest $end$ value
4      $matches1 \leftarrow \{C_Y | Y \in nodes\}$
5      $SP[p_i] \leftarrow false$
6      if (onSamePath($matches1$)) then
7          $SP[p_i] \leftarrow true$
8      else
9          if ($\exists Y \in nodes : \neg$ empty($S_Y$)) then
10             $matches2 \leftarrow \{empty(S_Y)?C_Y : top(S_Y) | Y \in nodes\}$
11             if (onSamePath($matches2$)) then
12                 $SP[p_i] \leftarrow true$
13     if ($\neg SP[p_i]$) then
14         advanceUntilSP($nodes$)
15         if ($noMoreSolns$) then
16             return
17     for (every child node $Y$ of $X$ annotated by $p_i$) do
18         $SP_Y[X, p_i] \leftarrow SP[p_i]?\ C_{node} : null$
```

**Function onSamePath(**$matches$**)**

```
1  $minEnd \leftarrow \min_{m \in matches}\{m.end\}$
2  $maxStart \leftarrow \max_{m \in matches}\{m.start\}$
3  return ($maxStart \leq minEnd$)
```

**Procedure advanceUntilSP(**$nodes$**)**

```
1  repeat
2      $minENode \leftarrow$ minarg$_{Y \in nodes}\{C_Y.end\}$
3      advance($minENode$)
4      if (eos($minENode$)) then
5          $noMoreSolns \leftarrow true$
6      $matches \leftarrow \{C_Y | Y \in nodes\}$
7  until ($noMoreSolns \vee$ onSamePath($matches$))
```

participates in a solution of $Q$, this match should be returned by $getNext$ before the match of $X$ in the same solution is returned. In order to enforce this returning order, we let $getNext$ "jump" to and continue its traversal from an ancestor of $X$ before $X$ is returned (lines 19-20 in $getNext$). The target ancestor node of $X$ is chosen as shown in the example below: consider again the dag of Figure 2. The path from the root $R$ to $X$ in bold denotes the invocation path of $getNext$ from $R$ to $X$. The invocation path is recorded in an array $invPath$ associated with each query node (line 4). Assume $P_1$ is the node under consideration by $getNext$, and $P_1$ has no solution in $\beta$. Assume also that $P_2$ has not yet been returned by $getNext$ but has a solution in $\beta$. Function $getNext$ on $P_1$ will return the *lowest ancestor* of $P_2$ among the nodes of $invPath[P_1]$ (which is node $W$). This enforces $getNext$ to go upwards along the invocation path of $P_1$ until it reaches $W$. From there, $getNext$ continues its traversal on the next child $V$ of $W$.

The same technique is also used when there is an unvisited node $Z$ annotated by a partial path that also annotates $X$, but the current match $C_Z$ of $Z$ has a smaller $start$ value than $C_X$. The existence of such a node is detected using the sink nodes of $Q$ (lines 21-22). This technique ensures that the matches of nodes in a same partial path are returned by $getNext$ in the order of their $start$ value.

**Dealing with the same-path constraint.** Let $X$ denote the node currently under consideration by $getNext$. After $getNext$ finishes its traversal of the subdag $Q_X$ and comes back to $X$, it invokes procedure $updateSPStatus$ (lines 12 and 25). Procedure $update$-$SPStatus$ (shown in Listing 2) checks the satisfaction of the same-path constraints for the subdag $Q_X$, and updates the data structures $SP$ and $SP_Y$ (described below) accordingly to reflect the result of the check.

Data structure $SP$ is a boolean array indexed by the set of partial paths annotating $X$ in the query $Q$. For each partial path $p_i$, $SP[p_i]$ indicates whether the same-path constraint for $p_i$ in $Q_X$ is satisfied by the matches of nodes in $Q_X$ (i.e., whether the matches of the nodes that are below $X$ and are annotated by $p_i$ in $Q$ lie on the same path in the XML tree). Let $nodes$ denote the sink nodes of $p_i$ in $Q_X$ (line 2 in $update$-$SPStatus$). In order to check the same-path constraint for $p_i$, it is sufficient to check whether the matches of sink nodes in $nodes$ lie on the same path in the XML tree. Note that the match of a sink node can be its current match or the one that has already been returned by $getNext$ and is now in its stack.

Procedure $updateSPStatus$ uses function $onSamePath$ to check if the matches of a set of query nodes lie on the same path in the XML tree (lines 6 and 11). This check is based on Proposition 1. If the same-path constraint is not satisfied, procedure $advanceUntilSP$ is invoked to advance the cursors of the nodes in $nodes$ until the current matches of the nodes lie on the same path in the XML tree or one of the cursors reaches the end of its list. In the latter case, it is guaranteed that there are no new solutions for $Q$. Hence, a boolean flag $noMoreSolns$ is set to $false$ in order for $PartialTreeStack$ to end the evaluation (line 5 in $advanceUntilSP$). During each iteration in $advanceUntilSP$, the node in $nodes$ whose current match has the smallest $end$ value is chosen and its cursor is advanced (lines 2-3). This way of advancing the cursors guarantees that all the matches of the nodes in $nodes$ that satisfy the same-path constraint will be eventually detected.

Figure 3 shows an example of cursor movement during evaluation that results in the current matches of the sink nodes of a query to lie on the same path.

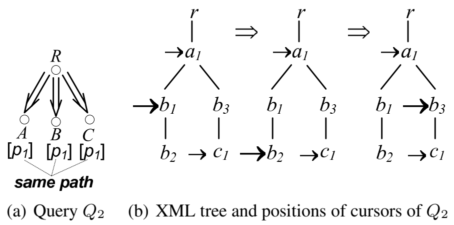


**Fig. 3.** A sequence of cursor movements resulting in the current matches of sink nodes $A$, $B$ and $C$ of $Q_2$ to lie on the same path

Every non-root query node $Y$ in $Q$ is associated with a two-dimensional array $SP_Y$. The first dimension of $SP_Y$ is indexed by the parents of $Y$ in $Q$, while the second one is indexed by the partial paths annotating $Y$ in $Q$. For every parent $X$ of $Y$ and partial path $p_i$, if the same-path constraint for $p_i$ in $Q_X$ is satisfied, $SP_Y[X, p_i]$ stores the current match of $node$ (line 18 in $updateSPStatus$). $node$ denotes the sink node of $p_i$ in the subdag $Q_X$ whose current match has the smallest $end$ value (line 3). Otherwise, $SP_Y[X, p_i]$ is set to $null$ (line 18). Note that $node$ is not necessarily a node in $Q_Y$ but can be a node in the subdag rooted at a sibling of $Y$ under the common parent $X$. Array $SP_Y$ is updated by procedure $updateSPStatus$ when the parent $X$ of $Y$ is under consideration by $getNext$, and $Y$ has a solution in the current binding of $Q$.

Array $SP_Y$ records the execution states that are needed to prevent redundant computations of $getNext$. For a selected node $Y$, the non-$null$ values of $SP_Y$ indicate that node $Y$ has a solution in the current binding of $Q$ and should be returned by $getNext$ (lines 23-24 in $getNext$). In this case, no call to procedure $updateSPStatus$ is needed.

**Main Algorithm.** The main part of $PartialTreeStack$ repeatedly calls $getNext(R)$ to identify the next candidate node for processing. For a selected node $X$, $PartialTreeStack$ removes from some stacks entries that are not ancestors of $C_X$ in the XML tree. The cleaned stacks are: (1) the stack of $X$, (2) the parent stacks of $X$, and (3) the stacks of sink nodes of every partial path of which $X$ is a sink node. Subsequently, $PartialTreeStack$ checks if for every parent $P$ of $X$, the top entry of stack $S_P$ and $C_X$ satisfy the structural relationship between $P$ and $X$ in the query. If this is the case, we say that $C_X$ has *ancestor extensions*. Then, $PartialTreeStack$ creates a new entry for $C_X$ and pushes it on $S_X$.

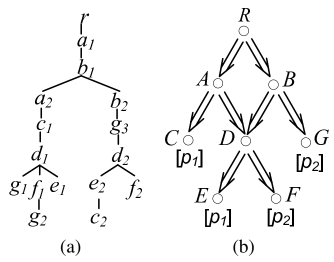
If $X$ is a sink node of a partial path $p_i$ and the stacks of all the sink nodes of $p_i$ are non-empty, it is guaranteed that the stacks contain at least one solution of $p_i$. Subsequently, a procedure is invoked to iteratively output all the solutions of $p_i$ that involve $C_X$. Such a procedure can be found in [20].

Finally, all the previously generated partial path solutions are merge-joined to produce the answer of the query. In the interest of space, the details of the main part are omitted.

## 5.2 An Example

We evaluate query $Q_3$ of Figure 4(b) on the XML tree of Figure 4(a) using Algorithm $PartialTreeStack$. The answer is shown in Figure 4(c). In Figure 5 we show different snapshots of the query stacks during the execution of the algorithm. Initially, the cursors of $R$, $A$, $B$, $D$, $C$, $E$, $G$, and $F$ are at $r$, $a_1$, $b_1$, $d_1$, $c_1$, $e_1$, $g_1$, and $f_1$, respectively. Before the first call of $getNext(R)$ returns $r$, $g_1$ is discarded by $advanceUntilSP$ because $g_1$ and $f_1$ are not on the same path.

Right after the eighth call returns $e_1$, the stacks contain solutions for the partial path $p_1$, and are produced by $outputPPSolutions$ (Figure 5(a)). At this time, the cursors of $R$, $A$, $B$, $D$, $C$, $E$, $G$, and $F$ are at $\infty$, $\infty$, $b_2$, $d_2$, $c_2$, $e_2$, $g_2$, and $f_2$ respectively. In the next call, $getNext$ first goes up from $D$ to $R$, then continues on $B$ because $b_2$ is the ancestor of $d_2$. This call finally returns $g_2$ since $g_2.start < d_2.start$. Subsequently, the solutions for the partial path $p_2$ are produced (Figure 5(b)). The eleventh call returns $g_3$ instead of $d_2$ because $g_3.start < d_2.start$. After $f_2$ and $c_2$ are returned, the solutions for $p_2$ and $p_1$ are generated respectively in that order (Figure 5(c)). Finally, these partial path solutions are merge-joined to form the answer of $Q_3$ (Figure 4(c)).



| R | A | B | D | C | E | G | F |
|---|---|---|---|---|---|---|---|
| $r$ | $a_1$ | $b_1$ | $d_1$ | $c_1$ | $e_1$ | $g_2$ | $f_1$ |
| $r$ | $a_1$ | $b_1$ | $d_2$ | $c_2$ | $e_2$ | $g_3$ | $f_2$ |
| $r$ | $a_1$ | $b_2$ | $d_2$ | $c_2$ | $e_2$ | $g_3$ | $f_2$ |
| $r$ | $a_2$ | $b_1$ | $d_1$ | $c_1$ | $e_1$ | $g_2$ | $f_1$ |

(c)

**Fig. 4.** (a) An XML tree $T$, (b) Query $Q_3$, (c) the answer of $Q_3$ on $T$

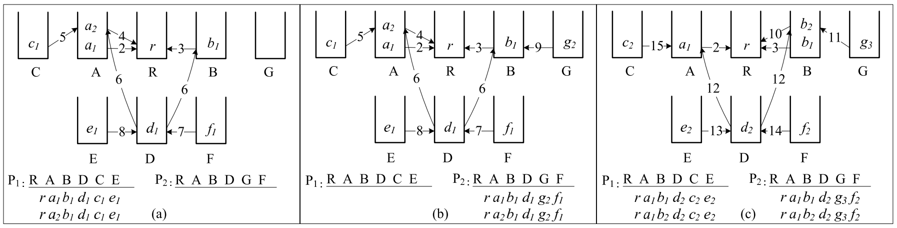


**Fig. 5.** Three snapshots of the execution of $PartialTreeStack$ on query $Q_3$ and the XML tree $T$ of Figure 4 (the numbers labeling the pointers denote the call to $getNext(R)$ as a result of which these pointers were created)

## 5.3 Analysis of PartialTreeStack

**Correctness.** Assuming that all the structural relationships in a PTPQ $Q$ are regarded as descendant, whenever a node $X$ is returned by $getNext(R)$, it is guaranteed that the current match $C_X$ of $X$ participates in a solution of subdag $Q_X$. These solutions of $Q_X$ constitute of a superset of its solutions appearing in the answer of $Q$. Moreover, $getNext(R)$ always returns a match before other descendant matches of it in a solution of $Q$. In the main part of $PartialTreeStack$, $C_X$ is pushed on $S_X$ iff $C_X$ has ancestor extensions. Whenever $C_X$ is popped out of its stack, all the solutions involving $C_X$ have been produced. Based on these observations, we can show the following proposition.

**Proposition 4.** *Given a PTPQ $Q$ and an XML tree $T$, algorithm $PartialTreeStack$ correctly computes the answer of $Q$ on $T$.*

**Complexity.** Given a PTPQ $Q$ and an XML tree $T$, let $|Q|$ denote the size of the query dag, $N$ denote the number of query nodes of $Q$, $P$ denote the number of partial paths of $Q$, $IN$ denote the total size of the input lists, and $OUT$ denote the size of the answer of $Q$ on $T$. The *ancestor dag* of a node $X$ in $Q$ is the subdag of $Q$ consisting of $X$ and its ancestor nodes. In [21], the *recursion depth* of $X$ of $Q$ in $T$ is defined as the maximum number of nodes in a path of $T$ that are images of $X$ under an embedding of the ancestor dag of $X$ to $T$. We define the recursion depth of $Q$ in $T$, denoted by $D$, as the maximum of the recursion depths of the query nodes of $Q$ in $T$.

**Theorem 1.** *The space usage of Algorithm $PartialTreeStack$ is $O(|Q| \times D)$.*

The proof follows from the fact that: (1) the number of entries in each stack at any time is bounded by $D$, and (2) for each stack entry, the size of $ptrs$ is bounded by the out-degree of the corresponding query node.

When $Q$ has no child structural relationships, Algorithm $PartialTreeStack$ ensures that each solution produced for a partial path is guaranteed to participate in the answer of $Q$. Therefore, no intermediate solutions are produced. Consequently, the CPU time of $PartialTreeStack$ is independent of the size of solutions of any partial path in a descendant-only PTPQ query.

The CPU time of $PartialTreeStack$ consists of two parts: one for processing input lists, and another for producing the query answer. Since each node in an input list is accessed only once, the CPU time for processing the input is calculated by bounding the time interval between two consecutive cursor movements. The time interval is dominated by updating array $SP_X$ for every node $X$ and is $O(|Q| \times P)$. The CPU time on generating partial path solutions and merge-joining them to produce the query answer is $O((IN + OUT) \times N)$.

**Theorem 2.** *Given a PTPQ $Q$ without child structural relationships and an XML tree $T$, the CPU time of algorithm $PartialTreeStack$ is $O(IN \times |Q| \times P + OUT \times N)$.*

Clearly, if the size of the query is insignificant compared to the size of data, $Partial$-$TreeStack$ is asymptotically optimal for queries without child structural relationships.

## 6 Experimental Evaluation

We ran a comprehensive set of experiments to assess the performance of *PartialTreeStack*. In this section, we report on its experimental evaluation.

### 6.1 Comparison Algorithms

As mentioned ealier, no previous algorithms exist in the inverted list model for the class of PTPQs. In order to assess the performance of *PartialTreeStack*, we designed, for comparison, two approaches that exploit existing techniques for more restricted classes of queries. The first approach, called $TPQGen$, is based on Proposition 2. Given a PTPQ $Q$, $TPQGen$: (1) generates a set of TPQs which is equivalent to $Q$, (2) uses the state-of-the-art algorithm [10] to evaluate them, and (3) unions the results to produce the answer of $Q$.

The second approach, called $PartialPathJoin$, is based on decomposing the given PTPQ into a set of queries corresponding to the partial paths of the PTPQ (*partial path queries*). For instance, for the PTPQ $Q_1$ of Figure 1(a), the partial path queries corresponding to the partial paths $p_1$, $p_2$, and $p_3$ of Figure 1(b) are produced. Given a PTPQ $Q$, $PartialPathJoin$: (1) uses the state-of-the-art algorithm [20] to evaluate the corresponding partial path queries, and (2) merge-joins the results on the common nodes (nodes participating in the node sharing expressions) to produce the answer of the PTPQ.

### 6.2 Experimental Results

**Setup.** We ran our experiments on both real and synthetic datasets. As a real dataset, we used the $Treebank$ XML document [1]. This dataset consists of around 2.5 million nodes and its maximum depth is 36. It includes deep recursive structures. The synthetic dataset is a set of random XML trees generated by IBM's XML Generator [2]. This dataset consists of 1.5 million nodes and its maximum depth is 16. For each measurement on the synthetic dataset, 10 different XML trees were used. Each value displayed in the plots is averaged over these 10 measurements.

On each of the two datasets, we tested the 4 PTPQs shown in Figure 6. Our query set comprises a full spectrum of PTPQs, from a simple TPQ to complex dags. The query labels are appropriately selected for the $Treebank$ dataset, so that they can all produce results. Thus, node labels $R$, $A$, $B$, $C$,

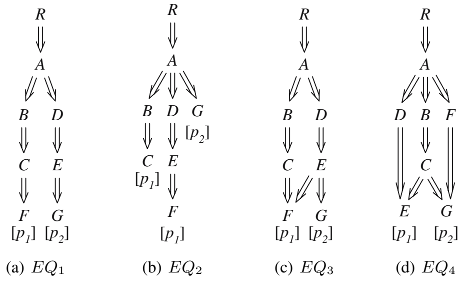
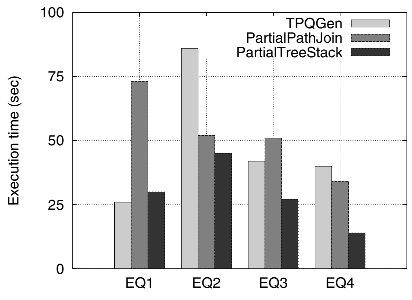


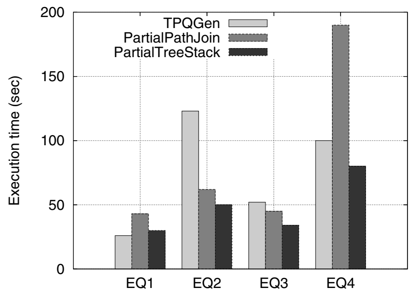
**Fig. 6.** Queries used in the experiments

[1] www.cis.upenn.edu/∼treebank

[2] www.alphaworks.ibm.com/tech/xmlgenerator



(a) Execution time (Treebank)



(b) Execution time (Synthetic data)

**Fig. 7.** Evaluation of PTPQs on the two datasets

$D$, $E$, $F$ and $G$ correspond to $FILE$, $EMPTY$, $S$, $VP$, $SBAR$, $PP$, $NP$ and $PRP$, respectively, on $Treebank$.

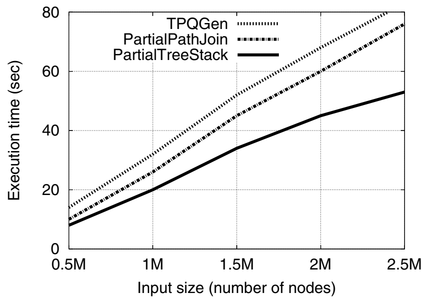
We implemented all algorithms in C++, and ran our experiments on a dedicated Linux PC (Core 2 Duo $3GHz$) with $2GB$ of RAM.

**Query execution time.** We compare the execution time of *TPQGen*, *PartialPathJoin* and *PartialTreeStack* for evaluating the queries in Figure 6 over the two datasets. Figures 7(a) and 7(b) present the evaluation results. As we can see, *PartialTreeStack* has the best time performance, and in most cases it outperforms either *TPQGen* or *PartialPathJoin* by a factor almost 2. Its performance is stable, and does not degrade on more complex queries and on data with highly recursive structures.

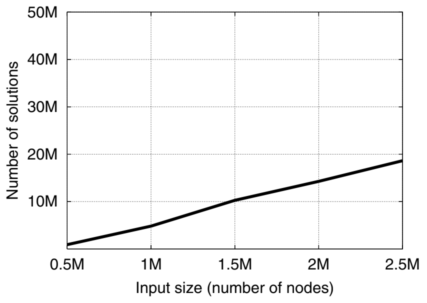
The execution time of *TPQGen* is high for queries with a large number of TPQs, for example, $EQ_2$. Query $EQ_2$ is equivalent to 10 TPQs. *TPQGen* shows the worst performance when evaluating $EQ_2$ on both datasets (Figure 7(a) and 7(b)).

*PartialPathJoin* finds solutions for each partial path of the query independently. It is likely that some of the partial path solutions do not participate in the final query answer (intermediate solutions). The existence of intermediate solutions affects negatively the performance of *PartialPathJoin*. For example, when evaluating $EQ_4$ on the synthetic data, *PartialPathJoin* shows the worst performance (Figure 7(b)), due to the large amount of intermediate solutions generated.

**Execution time varying the input size.** We compare the execution time of the three algorithms as the size of the input dataset increases. Figure 8(a) reports on the execution time of the algorithms increasing the size of synthetic dataset for query $EQ_3$. *PartialTreeStack* consistently has the best performance. Figure 8(b) presents the number of solutions of $EQ_3$ increasing the size of the dataset. As we can see, an increase in the input size results in an increase in the output size (number of solutions). When the input and the output size go up, the execution time of the algorithms increases. This confirms the complexity results that show dependency of the execution time on the input and output size. However, the increase in the execution time of *TPQGen* and *PartialPathJoin* is sharper than that of *PartialTreeStack*. The reason is that *PartialPathJoin* is also affected by the increase in the number of the intermediate solutions, while the performance of *TPQGen* is affected by the evaluation of 6 TPQs equivalent to $EQ_3$.

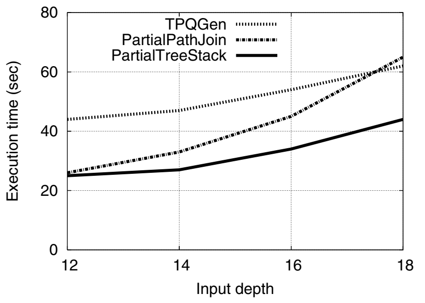


(a) Execution time

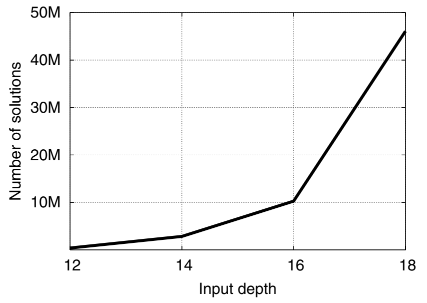


(b) Number of solutions

**Fig. 8.** Evaluation of $EQ_3$ on synthetic data with increasing size



(a) Execution time



(b) Number of solutions

**Fig. 9.** Evaluation of $EQ_3$ on synthetic data with increasing depth

**Execution time varying the input depth.** We also compare the execution time of the three algorithms as the depth of the input dataset increases. Figure 9(a) reports on the execution time of the algorithms increasing the input depth of synthetic dataset (its size is fixed to 1.5 million nodes) for query $EQ_3$. In all the cases, *PartialTreeStack* outperforms the other two algorithms. Figure 9(b) presents the number of solutions of $EQ_3$ increasing the input depth. As we can see, with the input depth increasing from $12$ to $18$, the output size increases from $0.4M$ to $46M$. When the output size goes up, the execution time of the algorithms increases. This again confirms our previous theoretical complexity results. We also observe that as the input depth increases, the execution time of *PartialTreeStack* increases very slowly. In contrast, the increase of the execution time of *PartialPathJoin* is sharper than that of the other two algorithms. The reason is that, for *PartialPathJoin*, an increase in the output size is accompanied by an increase in the number of intermediate solutions produced during evaluation. *TPQGen* does not increase sharper than *PartialPathJoin*. However, the execution time of *TPQGen* is strongly affected by the number of TPQs equivalent to the PTPQ, which in the worst case is exponential in the size of the PTPQ.

We also ran experiments to examine the impact of child relationships on the behavior of the algorithms. The results confirm that the presence of child relationships negatively affects the performance of *PartialTreeStack*, which nevertheless outperforms the other two algorithms in all the test cases. We omit these results here in the interest of space.

## 7 Conclusion

The motivation of this paper was the gap in the efficient evalution of broad fragments of XPath that go beyond TPQs. We considered PTPQs, a query language that generalizes and strictly contains TPQs. PTPQs can express a broad fragment of XPath. Because of their expressive power and flexibility, they are useful for querying XML documents whose structure is complex or not fully known to the user, and for integrating XML data sources with different structures.

We designed $PartialTreeStack$, an efficient stack-based holistic algorithm for PTPQs under the inverted lists evaluation model. To the best of our knowledge, no previous algorithms exist in the inverted list mode that can efficiently evaluate such a broad fragment of XPath. Under the reasonable assumption that the size of queries is not significant compared to the size of data, $PartialTreeStack$ is asymptotically optimal for PTPQs without child structural relationships. Our experimental results show that $PartialTreeStack$ can be used in practice on a wide range of queries and on large datasets with deep recursion. They also show that $PartialTreeStack$ largely outperforms other approaches that exploit existing techniques for more restricted classes of queries.

We are currently working on developing algorithms for the efficient computation of PTPQs using materialized views in the inverted lists evaluation model.

## References

1. World Wide Web Consortium site, W3C, http://www.w3.org/
2. Yu, C., Jagadish, H.V.: Querying complex structured databases. In: VLDB (2007)
3. Li, Y., Yu, C., Jagadish, H.V.: Schema-Free XQuery. In: VLDB (2004)
4. Theodoratos, D., Dalamagas, T., Koufopoulos, A., Gehani, N.: Semantic querying of tree-structured data sources using partially specified tree patterns. In: CIKM (2005)
5. Theodoratos, D., Wu, X.: Assigning semantics to partial tree-pattern queries. Data Knowl. Eng. (2007)
6. Peery, C., Wang, W., Marian, A., Nguyen, T.D.: Multi-dimensional search for personal information management systems. In: EDBT (2008)
7. Hristidis, V., Papakonstantinou, Y., Balmin, A.: Keyword proximity search on XML graphs. In: ICDE (2003)
8. Amer-Yahia, S., Lakshmanan, L.V.S., Pandit, S.: Flexpath: Flexible structure and full-text querying for xml. In: SIGMOD (2004)
9. Al-Khalifa, S., Jagadish, H.V., Patel, J.M., Wu, Y., Koudas, N., Srivastava, D.: Structural joins: A primitive for efficient XML query pattern matching. In: ICDE (2002)
10. Bruno, N., Koudas, N., Srivastava, D.: Holistic twig joins: optimal XML pattern matching. In: SIGMOD (2002)
11. Jiang, H., Wang, W., Lu, H., Yu, J.X.: Holistic twig joins on indexed XML documents. In: VLDB. (2003)
12. Wu, Y., Patel, J.M., Jagadish, H.V.: Structural join order selection for XML query optimization. In: ICDE (2003)
13. Lu, J., Chen, T., Ling, T.W.: Efficient processing of XML twig patterns with parent child edges: a look-ahead approach. In: CIKM (2004)
14. Chen, T., Lu, J., Ling, T.W.: On boosting holism in XML twig pattern matching using structural indexing techniques. In: SIGMOD (2005)

15. Jiang, H., Lu, H., Wang, W.: Efficient processing of XML twig queries with or-predicates. In: SIGMOD (2004)
16. Chen, L., Gupta, A., Kurul, M.E.: Stack-based algorithms for pattern matching on DAGs. In: VLDB (2005)
17. Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing XPath queries. ACM Trans. Database Syst. (2005)
18. Theodoratos, D., Placek, P., Dalamagas, T., Souldatos, S., Sellis, T.: Containment of partially specified tree-pattern queries in the presence of dimension graphs. VLDB Journal (2008)
19. Souldatos, S., Wu, X., Theodoratos, D., Dalamagas, T., Sellis, T.: Evaluation of partial path queries on XML data. In: CIKM (2007)
20. Wu, X., Souldatos, S., Theodoratos, D., Dalamagas, T., Sellis, T.: Efficient evaluation of generalized path pattern queries on XML data. In: WWW (2008)
21. Bar-Yossef, Z., Fontoura, M., Josifovski, V.: On the memory requirements of XPath evaluation over XML streams. In: PODS (2004)

# Mode Aware Stream Query Processing

Mingrui Wei and Elke Rundensteiner

Worcester Polytechnic Institute

**Abstract.** Many scientific applications including environmental monitoring, outpatient health care research, and wild life tracking require real-time stream processing. While state-of-the-art techniques for processing window-constrained stream queries tend to employ the delta result strategy (to react to each and every change of the stream sensor measurements), some scientific applications only require to produce results periodically - making the complete result strategy a better choice. In this work, we analyze the trade-offs between the delta and the complete result query evaluation strategies. We then design a solution for hopping window query processing based on the above analysis. In particular, we propose query operators equipped with the ability to accept either delta or complete results as input and to produce either as output. Unlike prior works, these flexible operators can then be integrated within one mode aware query plan - taking advantage of both processing methodologies. Third, we design a mode assignment algorithm to optimally assign the input and output modes for each operator in the mode aware query plan. Lastly, mode assignment is integrated with a cost-based plan optimizer. The proposed techniques have been implemented within the WPI stream query engine, called CAPE. Our experimental results demonstrate that our solution routinely outperforms the state-of-the-art single-mode solutions for various arrival rate and query plan shapes.

## 1 Introduction

A diversity of modern scientific applications ranging from environmental monitoring, outpatient health care research and wild life tracking require stream processing capabilities. To process unbounded data in real time, stream processing employs window predicates that restrict the number of tuples that must be processed for each stream at a time. One can move the windows over the input streams with the arrival of each new input tuple or periodically [1] [2], for example, every 5 minutes or every 100 tuples. We refer to the queries with the former kind of window movement as *sliding* window queries and with the later as *hopping* window queries.

In this paper, we focus on hopping windows for several reasons (with sliding being a special case with the smallest hop possible). First, the ability to display or notify the end user is physically limited by the hardware. It is unnecessary to update the results any faster than output hardware capabilities permit [3]. Second, even if the perfect device to display or notify the user at any speed was available, a user may not be able to respond at that rate. Third, even assuming the users were able to react to information at the speed of modern computers, the changes caused by one single tuple are very small, sometimes not noticeable by the human eye. For these reasons, the user may only choose to be notified periodically. Thus, hopping window queries can be commonly observed in many applications [1] [4] [2]. However, previous works typically neglect to deal with hopping window queries [5] [6] [7]. They implicitly process hopping window queries just like sliding window queries [1], hence ignoring the opportunity of designing processing techniques customized for hopping window queries.

In order to address the above problem, we analyzed the delta result approach (DRA) and the complete result approach (CRA). A complete result is a set of tuples that represents the query answer of the current window. A delta result is a set of tuples that represents the difference between the query answer of two consecutive windows. An operator that produces complete result is said to have a complete output mode (CO). If the input of the operator is a complete result, the operator is said to have a complete input mode (CI). If all the operators in a query have both complete input and output mode, the query is processing under CRA. Similarly, we can define delta result, delta input (DI), output (DO) mode and DRA.

In the case windows move forward one unit at a time (the sliding case), the update of the output is triggered by the arrival of each new tuple. Thus a large portion of the current window will be similar to the previous one. The same holds true for the result produced for the current window as well. Thus only a small amount of tuples indicating changes needs to be sent out to the next operator. However, in the case that windows move forward a large step at a time (the hopping case), most tuples that belonged to the previous window would no longer be valid for the current one. As a consequence, lots of new tuples would be produced when DRA is employed.

The question now arises if we can devise a more suitable processing strategy than maintaining the operators' states and the query output one tuple at a time. Let's consider the complete result method. Each operator receives the complete set of tuples belonging to the current windows from its upstream operators in the query plan. Thus it can produce the complete set of tuples as query result for the complete window for its downstream operators. In this scenario, neither state nor query result maintenance is needed. The possible concern now would be that tuples belonging to the overlapping part of two consecutive windows are being processing repeatedly.

From the above comparison, we can see that neither method guarantees to achieve the most efficient processing across different scenarios. Thus, we propose the notion of a mode-aware query plan that utilizes both the complete result and the delta result as intermediate representation between operators in different branches of the query plan. In our mode-aware query plan, each operator can work in either of the four possible input and output mode combinations independently of its predecessor or successor operators. The decision of which mode combination to assign to an operator depends on factors such as the size of the hop, the size of the window, the distribution of the data and the cost of the query operators within the plan. We develop an algorithm, called mode assignment, to assign the optimal mode configuration to each operator in a given plan driven by a cost model. We also design a query plan optimizer that considers mode assignment as part of cost-based optimization.

Contributions of this paper include:

- We study the trade-off between using the DRA and CRA in hopping window query processing. We then propose to seamlessly incorporate these two approaches into a single query plan to reduce system processing costs.
- We design the set of core stream relational query operators to equip them with dual input and output mode processing capabilities.
- Our mode assignment algorithm configures each operator within a given query plan to run in the best input and output mode combination to produce the optimal mode assignment. Our mode assignment algorithm is proven to be optimal yet efficient. Time complexity is only $O(n)$, where $n$ is the number of operators in the query plan.
- We extend conventional cost-based query optimization to consider the operator mode assignment problem together with operator positioning to produce an optimal mode-aware query plan.
- In our experimental study, our mode-aware query plans significantly outperform the single-mode plans generated by the state-of-the-art data stream processing solutions.

In the remainder of this paper, Section 2 describes the preliminary material. Section 3 compares the pros and cons of DRA and CRA. Section 4 describes the physical design of mode aware operators and presents the cost model. Section 5 discusses mode assignment, query plan optimization and rewriting heuristics. Section 6 shows experimental results. Section 7 consists of related work, while Section 8 concludes the paper.

## 2 PRELIMINARIES

### 2.1 Hopping Window Query Semantics

**Window And Hop.** In this paper, we use a CQL-like [1] syntax to specify queries. In particular, we use the **RANGE** clause to express the size of the window and the **SLIDE** clause to express the distance between two windows.

```
SELECT Istream (*)
FROM PosSpeedStr [RANGE 2 minutes SLIDE 1 minute]
```

In a hopping-window query above with SLIDE $h$ over $n$ input data streams, $S_1$ to $S_n$, assume we start processing at time $T = 0$, then the windows will move forward at time $T = 0+h, 0+2h, etc.$ Each input stream $S_j$ is assigned a window of size $w_j$. At time instance $T$ when the windows over the streams move forward by an interval of $h$ time units, the answer to the sliding-window query is equal

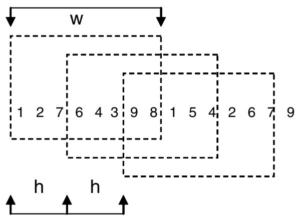


**Fig. 1.** Window size and Hop Size

to the answer of the conventional relational query whose inputs are the elements in the current window for each input stream. Figure 1 shows the relationship between windows and hops. We can see that the boundaries of each window are exactly $h$ time units from the previous windows respectively for each stream. In our discussion, we henceforth focus on the most commonly used window type, namely time-based windows [7] [4] [8] [9], though count-based windows could be processed in a similar manner.

**Handling Timestamps For Tuples.** Tuples from the input streams, $S_1$ to $S_n$, are timestamped upon their arrival at our system. A tuple $t$ is assigned two timestamps, $t$'s arrival time $t.ts$ and $t$'s expected expiration time, $t.Ets$. For a base tuple $t$ belonging to an input stream $S_i$, $t.Ets = t.ts + w_i$ is assigned to the tuple upon its arrival into the system. Thus at time T, the current window for stream $S_i$ contains the tuples that have arrived in between times $T - w_i$ and $T$. Tuples arriving between times $(T - w_i) - h$ and $T - h$ belong to the previous window. Operators in the query plan handle the timestamps of the input and output tuples based on the operator's semantics [7]. However, while all tuples logically expire latest at $T = Ets$, they may expire before $Ets$. We refer to such expiration as premature expiration. An example of an operator that causes such premature expiration is the difference operator.

## 2.2 Operator Classification

Continuous query operators can be divided into the following categories [5]:

- **Monotonic operators** produce an append-only output stream and therefore do not incur deletions from their answer set.
- **Weakest non-monotonic (WKS) operators** do not reorder incoming tuples during processing. Rather tuples expire in the same order in which they are generated by the operator (FIFO). Tuples are either dropped or appended to the output stream immediately.
- **Weak non-monotonic (WK) operators** may reorder incoming tuples during processing. Tuples may not expire result tuples in a FIFO order, but their expiration time can be determined by their $Ets$.
- **Strict non-monotonic (STR) operator** may expire tuples at unpredictable times, i.e., the expiration time of tuples produced by STR operators is not guaranteed to be determined solely by their $Ets$. Instead a special signal is needed to indicate expiration.

Most interesting operators in continuous query processing are at least WKS because windows have to applied in order to process infinite stream in finite memory. Thus, we will not discuss monotonic operator in this paper. The above classification initially introduced for sliding window queries in [5] also holds true for hopping window queries because the movement of the window itself does not reorder input tuples if DRA is employed.

However, the above classification is no longer applicable as is when CRA is utilized. Recall that CRA sends out the complete result for each window. Thus every tuple expires after the current window. Tuples belonging to the next window will be reproduced. Although *Ets* will determine how many windows a tuple belongs to, it does not determine when the tuple expires. Thus, when CRA is utilized, every operator falls into the same category. We call this new category **less strict non-monotonic (LSTR)**. LSTR operator expires tuples at a predictable time, but the expiration time of tuples cannot determined solely by their *Ets*.

## 3 Comparison of DRA Versus CRA Query Processing

**Delta Result Approach (DRA).** DRA can transmit only positive tuples through the query plan if the expiration times of all tuples can be determined via their *Ets*. The operator learns about the current time T by reading the newest positive tuple's timestamp *ts*. Then tuples with *Ets* smaller than T will be removed by the operator from its states. Such removal can be done locally within each operator.

However, some issues have to be considered for this approach. First, in order to quickly determine which tuples to expire, tuples in the operator states must be ordered by their expiration timestamps. However, a data structure supporting efficient expiration may not support operator semantics well. For example, while a hash table is a very efficient data structure for join operator states; there is no convenient way to order the tuples by their expiration timestamp. Also, a potentially unpredictable delay may arise due to no positive tuple reaching the root operator of the query plan if the lower level operators happen to be highly selective. Lastly, if the query plan contains a difference operator and other STR operators [5][7], premature expirations generated by these operators must be explicitly. In the remainder, we will refer to it as explicit deletion signal (EDS).

The above problems can be addressed if we use negative tuples[7] to signal both window semantics (via *Ets*) and operator semantics (EDS) expirations explicitly. In other words, negative tuples are responsible for undoing the effect of previously processed positive tuples. Negative tuples propagate through the query plan and are processed by operators in a similar way like positive tuples. However instead of causing insertions into operator states, they cause stateful operators to remove the corresponding positive tuples from their state. The downside is that twice as many tuples must be processed by the query plan because every tuple eventually expires and generates a corresponding negative tuple. In the case of high-volume streams, doubling the workload may adversely affect performance.

**Complete Result Approach (CRA).** While using CRA, operators do not need to store any tuples from previous windows. Therefore no state maintenance is needed either. Each operator directly uses the complete result input to compute its own complete result as output. CRA does not suffer from any of the above problems identified for DRA. However, when many tuples contained in two subsequent windows overlap, then CRA would not be utilizing the computation of the previously already computed result. In a situation where the result is more expensive to compute than to maintain, CRA may not be a suitable strategy.

From the above discussion, we can see that DRA captures the changes between two windows while CRA presents the complete picture. By default, the CQL syntax assumes the user wants the window to slide by the smallest time unit the system can offer. In such special case it is naturally efficient to indicate the changes between two consecutive windows compared to presenting the complete result of the current window. On the other hand, in the extreme cases when two windows do not overlap at all, all tuples from the previous windows will have to be removed from the states. In this case, maintaining the states on a per tuple-base as done by DRA is not desirable. In that cases, CRA can take full advantage of maintenance free processing while its deficiencies are not applicable here because little or no repeated computation is performed.

## 4 Mode Aware Physical Design

### 4.1 Conversion Operator

We design general purpose conversion operators that convert a delta result into a complete result and vice versa. Such conversion operator can be stand alone or integrate into the operator.

**Switch from Delta to Complete.** Let us first assume that the input of the conversion operator is **WKS**, which means every tuple expires in the order of its arrival. Then we design a linked list data structure that appends newly arriving tuples at one end and expires old tuples at the other end. Thus at the end of each hop, the conversion operator simply outputs its whole state. The cost for such conversion is a a small constant multiplied by the size of the input.

If the input of the conversion operator is **WK**, we will partition tuples by their *Ets*. Assume the current time is $T$. Then tuples with $T <= Ets < T + h$ belong to the first bucket. Tuples with $T + h <= Ets < T + 2h$ belong to the second bucket, and so on. Thus at the end of each hop, after outputting its current buckets, the conversion operator can simply discard the oldest bucket. Alternatively, if negative tuples are used to assist the expiration, we build a hashtable indexed on the key of the tuples. Thus, for every expiration, a quick hash look up is sufficient. In either case, the cost is linear in the size of the input. If the input of the conversion operator is **STR**, negative tuples are a must. Similar to WK, a hashtable can do the job efficiently. Thus the cost is the same as **WK**.

**Switch from Complete to Delta.** In order to support conversion from complete result to delta result, we have to be able to efficiently determine the difference between two sets of tuples. Hence, we will put the previous and the current result into two hashtables respectively. Conversion from complete to delta can be done in linear time in the size of the current complete result.

## 4.2 Operator Physical Design

In this subsection, we present the design of our mode aware query operators.

**Window Select, Window Project, Window union**. The select, project and union operators process tuples on the fly. There is no need for them to store any tuples as operator states. Thus they have no states to maintain and the choice of input and output mode makes no difference. They process tuples just like their sliding window counterparts.

**Window Join.** Nested loop or hash join can be used as join method for the natural join operator[9]. However, nested loop is most suitable for small window. Thus, we will not discuss the implementation of nested loop here. Instead, we use two hashtables as operator states to hold tuples from streams S and R respectively with the join attribute as the key to allow quick look-up for matched tuples. In the DIDO case, processing a tuple is done in the same way for both inputs. Insert, probe and purge have to be done on both sides thus join needs to access previous input tuples which still fall into the current window while processing the newly incoming tuples. Both positive and negative tuples will trigger output if matched tuples are found. However, output tuples triggered by negative tuples are negative. In the DICO case, instead of delta output, complete output needs to be generated. To achieve this semantics, as tuples arrive, the join operator keeps and maintains its previous output by purging out expired tuples, if any. All the new tuples are processed in the same way as in DIDO. Such maintenance is the only extra work compared to the DIDO case. In the CICO case, we know that the current output contains everything we need to know for the current window. Thus the operator can simply insert these tuples into the state, then probes and concatenates matching tuples. After that, the operator does not have to keep input tuples anymore. In the CIDO case, the operator needs extra work to convert complete input into delta output. To do that, previous outputted tuples are needed to be stored inside the operator. Current output will have to be compared with the previous output.

**Window Difference.** We use hashtables to hold tuples for the difference operator. A key of the hashtable correspond to the concatenation of all the attributes in the input tuples. In the DIDO case, as new tuples arrive, the difference operator has to maintain its states up-to-date by inserting positive tuples into the output and purging out expired and invalid tuples. For every new tuple, the difference operator determines whether to generate an output. In the DICO case, the difference produces its delta output first. In order to produce complete result, negative tuples in the delta result are used to remove invalid tuples from the previous complete result and positive tuples are added to the previous complete result. In the CICO case, the difference operator does not keep its previous input, as the input is complete. Input tuples will all be inserted into the states of the operator and perform the difference at once. In the CIDO cases, besides the states, the previous output is also needed for the conversion from complete to delta.

### 4.3 Cost Model for Operators

Each candidate plan is associated with a per-window cost. The cost includes inserting new tuples into the state, processing them, expiring old tuples and processing negative tuples if any. Such cost largely depends on operator's implementation. For each operator we define the following symbols in Table 1.

**Table 1.** Cost model symbols

| | |
|---|---|
| $\lambda_1$ | Left input rate |
| $\lambda_2$ | Right input rate (if the operator is unary, then $\lambda_2 = 0$) |
| $\lambda_o$ | Output rate |
| $D_i$ | number of tuples in delta input. $D_i = \lambda_i * h$ |
| $C_i$ | number of tuples in complete input. $C_i = \lambda_i * w_i$ |
| $s_i$ | size of the input $s_i = D_i, C_i$ |
| $s_o$ | size of the output |
| $\sigma$ | Selectivity |
| $c_{hash}$ | Hash constant |
| $w_i$ | window size |
| $\|W_i\|$ | state size $\|W_i\| = \lambda_i * w_i$ |
| $h$ | Slide distance |

Selection, projection and union process each tuple in constant time, therefore their cost is $\sum_i \lambda_i$. Their output rate is $\sigma_i \sum_i \lambda_i$. Union and projection has a selectivity of $\sigma_i = 1$. Hashtables are utilized as join states for all mode combinations. Thus in both DIDO and CICO cases the join cost for insert and lookup is $\sum_i s_i$. In the DIDO case, the cost for concatenation of the matched tuples is $|W_1|*D_2*\sigma+|W_2|*D_1*\sigma$. $|W_1|*D_2+|W_2|*D_1$ is cost for cross purge if we maintain states eagerly. In the CICO case, the cost for concatenation is $s_1 * s_2 * \sigma$. There is no state maintenace cost for the CICO case. In either case $\lambda_o$ is equal to the number of tuples concatenated divided by $min(w_1, w_2)$. For the difference operator, hashtables are used as operator states. The cost of difference is at least $c_{hash} \sum_i s_i$. In the DIDO case, the eager state maintenance cost is $|W_1| * D_2 + |W_2| * D_1$. In both cases $\lambda_o = (s_1 * \sigma)/w_1$. The cost for conversion from complete to delta and vice versa is the same as the corresponding conversion operator.

## 5 Mode-Aware Query Optimization

As mentioned earlier, the representation of intermediate results alone can greatly affect the execution cost of a plan. In this section, we discuss how to apply DRA and CRA in a integrated fashion within single mode aware query plan. Then we present several query plan optimization techniques for mode aware query plans.

### 5.1 Mode Assignment

A query plan $p$ is an ordered-tree structure composed of a collection of operators denoted as $OP$. We now design a mode assignment algorithm that assigns a mode combination to each operator *op* in $p$.

Mode constraint of operator *op* is represented by the tuple $< I, O >$ where:

– $I$ is a set of the modes of input streams.
– $O$ is a set of the modes of the output requirement.

$\delta$ is a mapping function that $\delta(op) : op \rightarrow M$. $M$ is a possible input and output mode combination. The domain of $M$ is $DIDO, DICO, CICO, CIDO$. The domain of $I$ is $DI, CI$. The domain of $O$ is $DO, CO$. . If there is no constraint on *op*, *op.I* and *op.O* each contains all the element in its domain. The root of a plan is an operator that communicates with the application such as a display device, a log file, even another query plan. Thus the $O$ of root *op* must conform to the constraints of the application. A query plan can have one or many leaf operators. These operators receive inputs for the plan from source streams. $I$ of the leaf operator corresponds to the mode of the incoming streams. Other operators in the query plan have no preset mode constraints. After associate with $\delta$, the operator is assigned a mode combination. Thus we also refer to $\delta(op)$ as an assignment of a operator. Once assigned, *op.I* and *op.O* remain fixed. If we associate each *op* in $p$ with a $\delta$, we attain an assignment $A(p)$ for query plan $p$. By default, we assume both input streams and output stream are in delta mode, thus all operators in a query plan initially have DIDO assigned as their mode combination.

We can also apply the assignment $A(p)$ to the subplan of $p$, $p_{sub}$. $A(p_{sub})$ is a subset of $A(p)$ that consists of mapping $\delta$ for operators in the subplan $p_{sub}$. An assignment $A(p)$ is said to be compatible with another assignment $A'(p)$ if and only if $O$ of $\delta$ for the root and $I$ of $\delta$ for each leaf operator of $p$ is the same as $A(p)$. In other words, externally, two assignments must have the same input and output mode. For each given plan $P$, we can obtain a set of assignments $\mathcal{A} = \{A(p), A'(p), \text{etc.}\}$. For each assignment $A(p)$, we can apply our cost model to obtain a cost $C(A(p))$. Such cost is the cost of executing the plan $p$ under the assignment $A(p)$. The mode assignment in $\mathcal{A}$ with minimum cost that conforms to the input mode and output mode requirements of the plan as a whole is called optimal mode assignment.

**Theorem 1 (Theorem of Mode Assignment Optimality).** *If a given plan $p$ has an optimal assignment $A(p)$, then for any sub-plan $p_{sub}$ of $p$, $A(p_{sub})$ is also optimal.*

*Proof (Proof of Mode Assignment Optimality).* For the sake of contradiction, suppose plan $p$ has an optimal assignment $A(p)$. But for a sub-plan $p_{sub}$, the assignment $A(p_{sub})$ is not optimal. Then for $p_{sub}$, choosing another compatible mode assignment $A'(p_{sub})$ can achieve a plan with lower cost while still satisfying the input and output requirements for $p_{sub}$. $C(A(p_{sub}))$ is larger than $C(A'(p_{sub}))$. If this is the case, we can substitute the assignment for plan $p_{sub}$

with the new assigned $A'$ by replacing $\delta(op)$ with $\delta'(op)$ to obtain a better assignment $A''(p)$. But this contradicts the assumption that the assignment $A(p)$ for plan $p$ is optimal. Thus, the assignment $A(p_{sub})$ must be optimal. □

If the query plan has neither input nor output requirements, a naive approach enumerating all possible combinations for each operator will take exponential time. Clearly brute force is not practical. Thus, we design a plan assignment algorithm that is guaranteed to produce optimal assignment for a given plan in $O(n)$, where $n$ is the number of operators in the given query plan. The pseudo code is shown in Algorithm 1. The mode assignment algorithm traverses the tree in postorder. It starts at the root of the plan and recursively determines the mode for its children. In the algorithm, *assignMode()* attach two assignments to the current: one with the least cost where $O = CO$ , and the other with the least cost where $O = DO$. Mode assign algorithm assigns mode directly if *op* is a leaf operator. Otherwise it will recursively get the mode of the left and right child of the current node. If two assignments for the same subplan are compatible, our algorithm always picks the one with the lower cost. By ensuring each subplan has an optimal assignment, as our algorithm traverses the tree once and an optimal assignment is found in $O(n)$ time.

**Algorithm 1.** assignMode (queryplan represented by root node)

```
1: if $op$ is a leaf node then
2:     assignMode($op$)
3: else
4:     for all $op_i \subset op.children$ do
5:         assognMode($op_i$)
6:     end for
7:     assignMode($op$)
8: end if
```

## 5.2 Optimal Mode Aware Plan Generation

Mode assignment alone can greatly improve the performance of a given plan. However, it cannot guarantee a truly optimal plan. Thus, we now incorporate the mode assignment into our cost-based plan search. In particular we employ a dynamic programming-based solution due to its popularity for query optimization [10] [11][12][13]. Other plan search algorithm could be equally employed. We briefly describe our algorithm while readers interested in dynamic programming can look at [10] [11][12] [13].

Our searchJoinOrder algorithm is shown in Algorithm 2. It considers all possible orderings to join the streams at the same time with mode assignment. First, it considers all two-way join plans using the streams as building blocks and calls the *joinPlans()* function to build a join plan $p$ from these building blocks. Then it applies $assignMode(p)$ to obtain the cost of producing complete result and delta result of $p$ respectively. From the two-way join plans and the streams, our searchJoinOrder algorithm then produces two three-way join plans with minimal costs of producing complete result and delta result. After that, it generates four-way join plans by considering all combinations of two two-way join plans

**Algorithm 2.** searchJoinOrder(Join Graph)

```
for $i = 2$ to $n$ do
    for all $S \subset R_1, ..., R_n$ such that $|S| = i$ do
        optPlan($S$)= $\oslash$
        for all $O \subset S$ do
            optPlan(S)=getMode( optPlan($S$) )$\cup$ getMode(joinPlans (optPlan($O$), optPlan($S \backslash O$)))
            prunePlans(optPlan(S))
        end for
    end for
end for
prunePlans($R_1, ..., R_n$)
```

**Algorithm 3.** placeDifferenceOperator(extended Join Graph)

```
Plans= $\oslash$
if RootGraph has difference source then
    for $i = 0$ to $n - 1$ do
        if $S_i$ is a difference source then
            for $i = 0$ to $n - 2$ do
                for all $S \subset S_0 ... S_{i-1}, S_{i+1} ... S_{n-1}$ such that $|S| = i$ do
                    Remove $S$ from RootGraph, ADD $S$ to LeftChild and RightChild of $S_i$
                    placeDifferenceOperator($S_i$ .LeftChild)
                    placeDifferenceOperator($S_i$ .RightChild)
                    Plans.AddPlan (searchJoinOrder(RootGraph))
                    Restore $S$
                end for
            end for
        end if
    end for
else
    Plans.AddPlan(searchJoinOrder(RootGraph))
end if
PickOptimalPlan(Plans)
```

and all combinations of a three-way join plan with a source stream and so on. The *prunePlans()* function discards unneeded plans. Pruning is possible because A ⋈ B and B ⋈ A produce semantically identical results. Thus we only store a plan with its assignment that produces DO with minimal cost and one that produces CO with minimal cost. They are retained in *optPlan* (A, B).

We further enhance our algorithm to consider difference operators as shown in Algorithm 3. Besides join ordering and mode assignment, placeDifferenceOperator algorithm considers all possible positions of the difference operators in query plan. The input of the algorithm is a plan with difference operators all the way pushed to the lowest possible level. The parent of a difference operator is an extended join graph. An extended join graph is a join graph where an internal node may be the source stream of a difference operator. The children of a difference operator are also extended join graphs. As the algorithm traverses the search space, it applies the rewrite: $S \bowtie (A - B) = (S \bowtie A) - (S \bowtie B)$. $S$ is a set of streams belonging to the parent of the difference operator. $A$ and $B$ are streams belonging to the left child and right child of the difference operator respectively. Our algorithm considers all possible cases to add $S$ to $A$ and $B$. It first considers cases where $|S| = 1$. At these cases, we remove one stream from the parent of the difference operator and add it to the left and right child. Then apply Algorithm 2 to the modified children and parent of the difference operator. After trying all possible cases with $|S| = 1$, we start with the original parent and children of the difference operator and try all possible cases with $|S| = 2$ and so on until $S$ contains every stream in the parent of the difference operator.

### 5.3 Heuristic Optimization

Optimal plan generation is not always practical due to the time constraint on stream optimization. In that case, we would consider a two step process, namely first heuristically optimize the plan shape then followed by our optimal mode assignment algorithm to improve the performance of query optimization. In this work, we first employ the following heuristic based on the difference pushdown rule. However, other heuristics could be chosen. The rule is that we always push the difference operator to the lowest level in a given query plan. Then we consider to elevate each difference operator one stream at a time, We will stop if all possible next elevations do not reduce the processing costs. In the case that the hop/window ratio is large this is likely to give us an optimal plan if incorporated with mode assignment. Other factors will also affect the effectiveness of this rule. For example, if the selectivity of the difference operator is low, the difference operator may greatly reduce the number of tuples that propagate through the query plan. If we consider all hop/window ratios and operator selectivities to be equally likely, pushing the difference operator down is an effective heuristic.

## 6 Experimental Evaluation

We have implemented the mode-aware query processing strategies for hopping queries within the WPI stream processing engine CAPE. Experiments were performed on two Linux machines. Each machine has 4 CPUs with 1000 MHz each and 4 GB of memory. We use one of the machines as stream generator and the other one as the processing engine. Our experiments are designed to:

– Validate the cost models by comparing the execution times estimated by our proposed cost model for various hopping window queries and the actual execution times produced by the prototype system;
– Examine the relative performance trends of the four different input and output combinations for different types of operator.
– Compare the performance gains achievable by assigning modes to a given query plan;
– Demonstrate the effectiveness of the difference push down heuristic under various parameter values, like selectivity, hop-window ratio, etc.
– Demonstrate the effectiveness of our solution of combining heuristic rule and mode assignment together compared to using either one of them in isolation.

We employ the CAPE data generator to generate synthetic stream data. Inputs to the data generator are the number of tuples, the number of attributes in the stream schema, the number of distinct values of each attribute in the schema, the stream rate (number of tuples per second) and its distribution. In our experiment, we use two distributions to verify our techniques work across

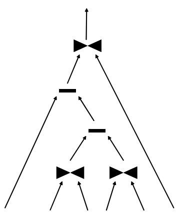


**Fig. 2.** Plan Shape for Query 3 and 4

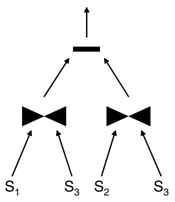


**Fig. 3.** Difference Operator Pull Up

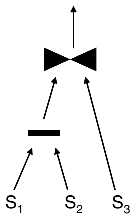


**Fig. 4.** Difference Operator Push Down

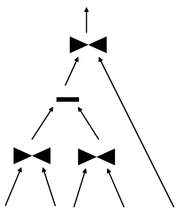


**Fig. 5.** Plan Shape for Query 5

different circumstances. One is Poisson distribution which is commonly use to model random events arriving at a system [7]. We also used pareto distribution to model a more busty arrival of data tuples. Each tuple is assigned a timestamp, whose value is determined based on the stream rate and its distribution. Mean interval between two tuples is 10 millisecond unless noted otherwise. For pareto distribution, we have $\frac{kx_{min}}{k-1} = 10$, where $x_{min} = 5$. Values of each attribute are assigned randomly using a uniform distribution.

### 6.1 Query 1: Single Join Operator Plan

We test 10 variations of Query 1 by varying the hop/window ratio from 10% to 100% by 10% each time for all 4 combinations of DIDO, DICO, CICO, and CIDO. The $y$ axis indicates time to complete each window. The $x$ axis is the hop/window ratio. The Figure 6 clearly shows that when ratio is small, the delta input is preferred by the Join operator. DIDO is the least expensive mode for this case because it does not conduct any redundant computation. DICO trails a little behind because of its overhead to convert incremental output into the complete output. CICO is the third candidate in performance because it has to compute lots of redundant tuples. CIDO is the most expensive strategy because not only does it have to compute redundant tuples, but also has to convert them into the delta result. On the other hand, we can see that when the ratio become larger and larger, CICO continuously gains ground and eventually surpasses DIDO. Because more and more tuples expire between two consecutive windows, thus the state maintenance cost goes up for delta inputs. Both the experimental run and the cost model show these trends. We conclude that when consecutive windows do not overlap at all, using CICO is 32% better than DIDO for a single join operator.

### 6.2 Query 2: Single Difference Operator Plan

We again test 10 variations of Query 2 by varying the hop/window ratio from 10% to 100% for all 4 combinations of DIDO, DICO, CICO, and CIDO. The $y$ axis is the time to complete each window, while the $x$ axis is the hop/window ratio. Figure 8 shows that although the delta input is still preferred by the Difference operator when the ratio is small, the gap between processing the

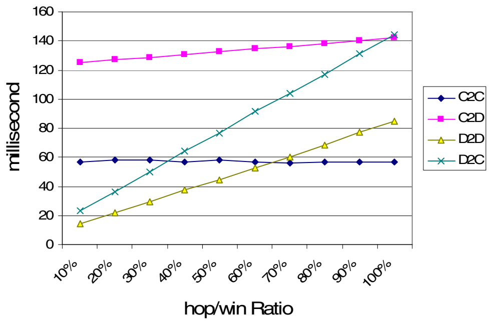


**Fig. 6.** Single Join Plan

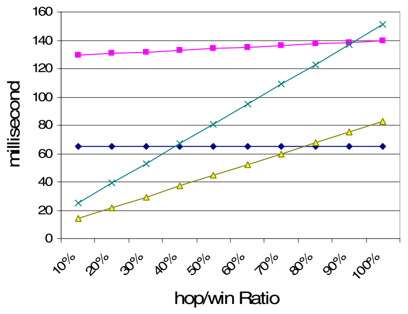


**Fig. 7.** Cost Model Single Join Plan

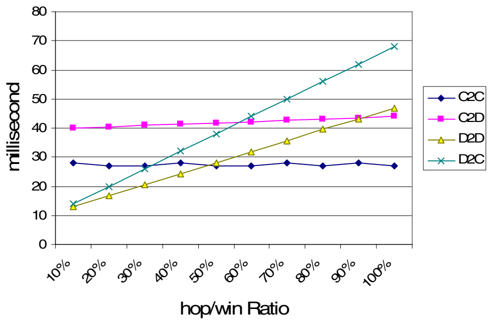


**Fig. 8.** Single Diff Plan

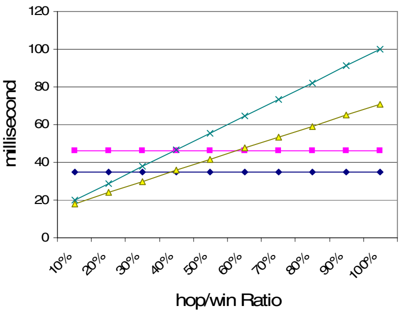


**Fig. 9.** Cost Model Single Diff Plan

delta input and the complete input is smaller than for the join counterpart. The trends are similar to the join operator, again confirmed by both the experimental run and the cost model estimations. When the windows do not overlap at all, using CICO can save up to 40% compared to DIDO in our setup.

## 6.3 Query 3: Assigned Plan Versus CICO Plan

In this experiment, we will demonstrate how running in a suitable mode can reduce cost of query processing, in particular, CRA to DRA when the hop/window ratio is small. The plan shape is shown in Fig 2. The $y$ axis indicates time used to complete each window. In this experiment, we tested two variations of Query 3. One is generic mode approach which uses CICO throughout the query plan. The other one is an assigned plan which internally uses DRA, while the input and output remain complete mode. Such assignment is obtained by running the mode assignment algorithm when the internal hop/window ratio equals 20%. The time to determine the mode is negatable. For each variation, we increase the hop/window ratio of the internal operators 20% at a time. The hop size is fixed at 5000 ms. We compare the actual running time of the CICO plan and the mode aware plan. In Figure 10, we can see that when the ratio is small, switching to delta internally is a good choice. But this assignment will not work

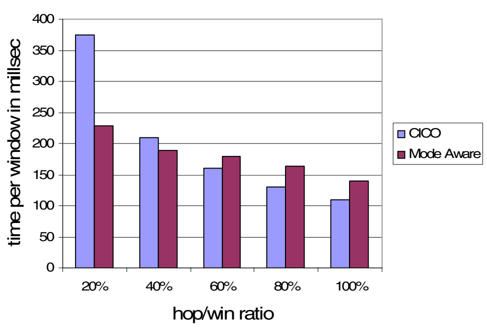


**Fig. 10.** Query 3: Mode Aware Plan

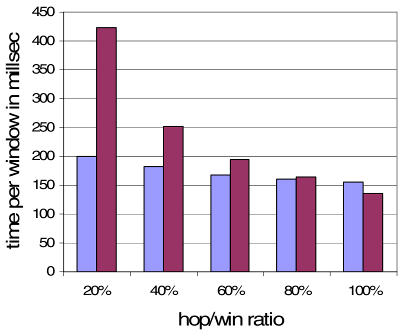


**Fig. 11.** Query 4: Mode Aware Plan

for when the ratio is large. Our mode assignment algorithm will know CICO is the best choice and will not chose the assigned plan which uses DRA internally.

## 6.4 Query 4: Assigned Plan Versus DIDO Plan

In this experiment, we will demonstrate the benefit of switching from DRA to CRA internally. The plan shape is shown in Fig 2. The $y$ axis indicates time to complete each window. In this experiment, we test two variations of Query 4. One is generic mode plan which uses DIDO throughout the query plan. The other one is an mode aware plan which internally uses CRA. Such assignment is obtained by running the mode assignment algorithm when the internal hop/window ratio equals 100%. For each variation, we increase the hop/window ratio of the internal operators in this experiment 20% at a time. The hop size is fixed at 5000 ms. We compare the actual running time of DIDO plan and the assigned plan. In Fig 11, we can see that as the ratio increases, the processing time of our mode aware plan decreases. When the ratio is at 100%, switching to complete internally is a good choice. When the ratio is small, our mode assignment algorithm will know DIDO is the best choice and will not chose the assigned plan which use CRA internally.

## 6.5 Heuristic Rule Evaluation

In this experiment, we randomly generate plan shapes with less than 8 operators. Then we randomly assign different selectivities to each operator. We use this as our input to both the heuristic optimizer and the cost-based query optimizer. Over 98% of the time, the plan returned by our heuristic was as good as the plan returned by the optimizer.

Figure 12 depicts a more detailed statistic of the difference operator push down plan compared to difference pull up plan. The plan shapes are shown in Fig 3 and Fig 4. In general, if each case has a equal possibility to arise, our heuristic rule works very well. However, if the query plan has a small hop/window ratio, and the size of the plan is large, pushing the difference down may not produce an optimal plan. That is because negative tuples generated by the difference

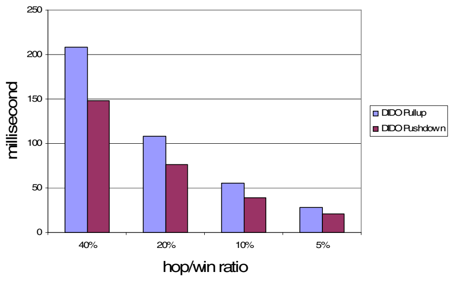


**Fig. 12.** Difference Operator Pullup versus Pushdown

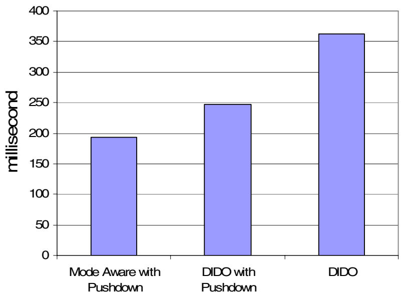


**Fig. 13.** Mode Aware Query Plan with Pushdown Heuristic

operator may affect a large number of operators in the query plan, potentially doubling the workload for those operators.

### 6.6 Putting It All Together

In this experiment, we compare the performance of three different plans for Query 5.

- The user default plan (DIDO)
- The plan with pushdown heuristic (DIDO with Pushdown)
- The mode assignment plan with heuristic (Mode Aware Plan with Pushdown)

The user default plan is a plan show in Fig 5 with delta input and delta output. The plan with push down heuristic is obtained by applying our heuristic optimizer on the default plan to push the difference operator down. The mode aware plan with push down heuristic is the plan obtained by running the mode assignment algorithm on the plan generated by the heuristic optimizer. In Fig 13, we can see that the user default plan has the worst performance. It has not taking advantage of the opportunity of the difference operator pushdown can reduce the amount of work the upstream join operator has to perform. The plan with pushdown heuristic has pushed the difference operator down, however, it is not as efficient as the mode aware plan with pushdown because the mode aware plan using the complete result between operators internally. In this scenario the upstream join operator does not have to maintain its states. By using the CRA, the stream engine can also avoid the overhead of processing negative tuples generated by the difference operator.

## 7 Related Work

Most stream query processing research over the past few years has focused on sliding window queries only. Cost models and optimization techniques have been developed [14][3]. However, they may not work well for hopping window queries because they all focus on minimizing the cost for processing individual tuple upon their arrival.

[6] presented several execution strategies for sliding window query processing with support for negative tuples. Their method supporting negative tuples focuses on tuples that expire maturely (fall out of window). It does not work well when larger amount of tuples expire prematurely. [7] further investigates incremental evaluation techniques over sliding windows with the aim to minimize overhead for handling negative tuples in the system. However, these techniques again are employed to capture the case when tuples expire maturely. Our work considers both the mature and premature expiration. We present a mode aware approach that combines DRA and CRA within one integrated query plan to attain minimize system overhead.

[15] presents a join reordering technique that works for not only natural joins but also antijoin. Their work is based on traditional relational databases thus does not consider the streaming environment nor the effect of negative tuples. [2] study the hopping window semantics and memory sharing of aggregate queries. Their techniques focus only on memory sharing of aggregate queries and cannot easily be transferred to queries with other operators. Lastly, our work is related to [5]. [5] has presented a classification of update patterns of continuous queries and applied it to solve two problems: 1) defining precise semantics of continuous queries with their update patterns, and 2) efficient query execution over sliding windows utilizing their update patterns. However their classification only applicable when using DRA. Due to varying ratio between hop size and window size, their processing method, data structure and heuristic rules do not work well.

## 8 Conclusion

In this paper, we addressed the problem of processing continuously hopping window queries. We designed mode aware operators which can work in any mode combination. We then proposed a cost-based mode assignment algorithm. It assigns a mode combination to each operator within a given query plan and guarantees optimality. We further enhance plan search optimizer with mode assignment algorithm and difference operator positioning. We develop analytical cost models for the above techniques and validate them through experiments. We also empirically studied the efficiencies of our difference operator push down heuristic and compared the case of different assignment and plan shapes with respect to stream statistics.

## References

1. Arasu, A., Babu, S., Widom, J.: The cql continuous query language: semantic foundations and query execution. VLDB J. 15(2), 121–142 (2006)
2. Krishnamurthy, S., Wu, C., Franklin, M.J.: On-the-fly sharing for streamed aggregation. In: SIGMOD Conference, pp. 623–634 (2006)
3. Mokbel, M.F., Xiong, X., Hammad, M.A., Aref, W.G.: Continuous query processing of spatio-temporal data streams in place. GeoInformatica 9(4), 343–365 (2005)

4. Li, J., Maier, D., Tufte, K., Papadimos, V., Tucker, P.A.: Semantics and evaluation techniques for window aggregates in data streams. In: SIGMOD Conference, pp. 311–322 (2005)
5. Golab, L., Ozsu, M.T.: Update-pattern-aware modeling and processing of continuous queries. In: ACM SIGMOD Conference, pp. 658–669 (2005)
6. Hammad, M.A., Aref, W.G., Franklin, M.J., et al.: Efficient execution of sliding-window queries over data streams. Technical Report 03-035, Purdue University (2003)
7. Ghanem, T.M., Hammad, M.A., Mokbel, M.F., Aref, W.G., Elmagarmid, A.K.: Incremental evaluation of sliding-window queries over data streams. IEEE Trans. Knowl. Data Eng. 19(1), 57–72 (2007)
8. Ghanem, T.M., Aref, W.G., Elmagarmid, A.K.: Exploiting predicate-window semantics over data streams. SIGMOD Record 35(1), 3–8 (2006)
9. Golab, L., Özsu, M.T.: Processing sliding window multi-joins in continuous queries over data streams. In: VLDB, pp. 500–511 (2003)
10. Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G.: Access path selection in a relational database management system. In: Bernstein, P.A. (ed.) Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, May 30 - June 1, pp. 23–34. ACM, New York (1979)
11. Ono, K., Lohman, G.M.: Measuring the complexity of join enumeration in query optimization. In: VLDB, pp. 314–325 (1990)
12. Vance, B., Maier, D.: Rapid bushy join-order optimization with cartesian products. In: SIGMOD Conference, pp. 35–46 (1996)
13. Kossmann, D., Stocker, K.: Iterative dynamic programming: a new class of query optimization algorithms. ACM Trans. Database Syst. 25(1), 43–82 (2000)
14. Babu, S., Motwani, R., Munagala, K., Nishizawa, I., Widom, J.: Adaptive ordering of pipelined stream filters. In: SIGMOD Conference, pp. 407–418 (2004)
15. Rao, J., Lindsay, B.G., Lohman, G.M., Pirahesh, H., Simmen, D.E.: Using eels, a practical approach to outerjoin and antijoin reordering. In: ICDE, pp. 585–594 (2001)

# Evaluating Reachability Queries over Path Collections

Panagiotis Bouros[1,⋆], Spiros Skiadopoulos[2], Theodore Dalamagas[3], Dimitris Sacharidis[1], and Timos Sellis[1,3]

[1] National Technical University of Athens
9 Iroon Polytechniou, Athens 157 80, Greece
{pbour,dsachar}@dblab.ntua.gr
[2] University of Peloponnese
Karaiskaki Street, Tripoli 221 00, Greece
spiros@uop.gr
[3] Institute for the Management of Information Systems — "Athena" R.C.
17 G. Mpakou, Athens 115 24, Greece
{dalamag,timos}@imis.athena-innovation.gr

**Abstract.** Several applications in areas such as biochemistry, GIS, involve storing and querying large volumes of sequential data stored as *path collections*. There is a number of interesting queries that can be posed on such data. This work focuses on reachability queries: given a path collection and two nodes $v_s$, $v_t$, determine whether a path from $v_s$ to $v_t$ exists and identify it. To answer these queries, the *path-first search* paradigm, which treats paths as first-class citizens, is proposed. To improve the performance of our techniques, two indexing structures that capture the reachability information of paths are introduced. Further, methods for updating a path collection and its indices are discussed. Finally, an extensive experimental evaluation verifies the advantages of our approach.

**Keywords:** path collections, reachability queries.

## 1 Introduction

Several applications in various areas involve storing and querying large volumes of sequential data. For instance, the metabolic networks in biochemistry applications deal with large collections of pathways, i.e., series of chemical reactions occurring within a cell [1]. Another example comes from Geographic Information Systems (GIS) and geodata services, where the recent advances in infrastructure, and the proliferation of earth observation applications (e.g., GPS technology), have resulted in the abundance geodata. Path collections are typical in web sites such as ShareMyRoutes.com, which archive popular touristic routes, i.e., sequences of waypoints or points of interest (POIs), uploaded by users.

⋆ The author is supported by the Greek State Scholarships Foundation (IKY).

These datasets share a common structure. They involve items that connect with each other to form sequences. In the sequel, we refer to items as nodes and to sequences as paths. There is a number of interesting queries that can be posed on path collections. The focus of this work is on reachability queries: given a path collection and two nodes $v_s$, $v_t$, determine whether a path from $v_s$ to $v_t$ exists and identify it. Note that this path need not be present in the collection, but constructed by combining parts of stored paths. We present two distinct application scenarios. Consider a collection of metabolic pathways. In this context, reachability queries answer whether there exists a cause-effect relationship between two chemical reactions in some pathway, and their intermediates (i.e., the reactants). Furthermore, consider an archive of popular touristic routes. Reachability queries answer whether there exists a meaningful/recommended route between two touristic attractions.

A path collection can be trivially mapped to a graph, where its nodes are those contained in the paths. Hence, reachability queries can be evaluated by standard techniques that fall in three categories: (i) search algorithms, e.g., depth-first search, (ii) methods based on the pre-computation of the graph's transitive closure (TC), or (iii) approaches that pre-process the graph to construct a reachability encoding scheme. These techniques share their strengths and weaknesses. Exploiting a search algorithm has minimum space requirements but in the worst case we need to examine all the edges of the graph to answer a query. Considering the TC of the graph uncompressed is very efficient as far as querying is concerned, but the complexity of the construction time and the space requirements make this solution infeasible in practice. Works like 2-hop labels [2] that compress the TC of the graph, or labelling schemes [3,4,5] have been proposed to encode the reachability information of the graph. These schemes determine whether there exists a path between two nodes and some of the schemes can also identify that path.

It is important to note that techniques of the last two categories require preprocessing and are only efficient for datasets that do not frequently change. In our setting, however, there are frequent path insertions in the collection dramatically modifying the associated graph and rendering the pre-processed data useless. Based on this observation, we introduce the *path-first search* (pfs) paradigm for evaluating reachability queries on path collections. Briefly the main idea is to examine entire paths at once rather than single nodes. We present an index structure, termed $\mathcal{P}$-*Index*, that provides efficient access to the paths and devise the pfsP algorithm, which utilizes it. Then, we present $\mathcal{H}$-*graph*, a novel graph-representation of a path collection that captures information about shared nodes among paths, construct an appropriate index, $\mathcal{H}$-*Index*, and introduce the pfsH algorithm. Furthermore, we present methods for updating the index structures when new paths arrive. Finally we present an extensive experimental study verifying the advantages offered by our methods.

**Outline**. Section 2 reviews the literature on evaluating reachability queries on graphs. Section 3 formally defines the problem of evaluating reachability over path collections. Section 4 introduces the pfs and pfsP algorithms, and $\mathcal{P}$-*Index*.

Section 5 introduces the graph-based model of $\mathcal{H}$-*graph*, defines $\mathcal{H}$-*Index* and presents the pfsH algorithm. Section 6 discusses index maintenance. Section 7 presents the experimental study and Section 8 concludes the paper.

## 2 Related Work

The simplest way to evaluate reachability queries is to traverse the graph at query time exploiting a search algorithm, e.g., depth-first search (dfs). This approach has minimum space requirements — we store only the adjacency lists of the graph. On the other hand, to answer a query, we need to search all the edges in the worst case. In our work, we propose a search method, in the spirit of dfs, that operates on the paths of a collection instead of the edges of a graph.

Another option is to pre-compute and store the transitive closure (TC) of the graph. Then, we can explore the encoding scheme in [6] to assign to each node $v$ a set of triples $\langle destination, via, label \rangle$. Entry "via" denotes the first hop in the path from $v$ to the destination node. At query time, we determine the existence of a path between two nodes by a single lookup on the encoding scheme and identify it performing a number of lookups. The problem with this approach lies in computing the TC of the graph. Efficient algorithms for computing the TC in relational databases have been proposed, e.g., [7]. Even so, the computation time of $O(|V|^3)$ and the space requirements of $O(|V|^2)$ prevent us from applying this solution especially for large graphs. In our work, we do not pre-process the path collections to compute all the possible transitions between the nodes. We exploit a graph-representation of the path collections to capture the possible transitions between the paths according to their common nodes.

To reduce the storage cost of the TC, Cohen et al. [2] propose 2-hop labels. They identify a subset of the nodes that best capture the reachability information of a graph. Thus, for each node $v$, they construct a list with part of the nodes that can reach $v$ ($L_{in}[v]$) and another one with part of the nodes reachable from $v$ ($L_{out}[v]$). This scheme requires $O(|V| \cdot \sqrt{|E|})$ space and can determine the existence of a path between two nodes $v_s$, $v_t$ by checking whether $L_{out}[v_s]$ and $L_{in}[v_t]$ have a common node, the so called center $v_{center}$. To identify the path from $v_s$ to $v_t$, we need to repeat the procedure for the paths from $v_s$ to $v_{center}$ and from $v_{center}$ to $v_t$. The problem with this approach lies in the construction cost. Computing the optimal 2-hop cover is NP-hard and requires the computation of the TC. Therefore after the introduction of 2-hop labels, a number of works proposed methods to avoid the computation of TC and to reduce the construction time, e.g., [8] and [9]. In our work, for each node $v$, we exploit the common nodes of the paths containing $v$ with the other paths of the collection to capture the connectivity information of $v$.

In the context of labelling schemes for graphs, [3] proposes an interval labelling scheme. Considering both the spanning tree of the graph, and the remaining edges, they assign to each node $v$ a sequence of intervals $L[v]$. In [5], Wang et al. introduce Dual-Labeling for sparse graphs. In [10], Trißl et al. introduce GRIPP scheme for large graphs. Finally, the idea in [4] is instead of constructing the spanning tree of the graph, to partition the graph into a set of paths $P$ and then create the so called path-tree cover $G[T]$. The path-tree cover is a graph formed by the paths of $P$ (as nodes) and the edges of the initial graph that are not included in any path. In our work we do not assign labels to the nodes of the collection for encoding their connectivity information. We index the paths that contain each node. The graph-based representation of a collection we present, called $\mathcal{H}$-*graph*, resembles the one proposed in [4]. However, in [4] each node is contained in exactly one path, whereas in our work a node can be included in several paths of the collection. In addition, the edges of $\mathcal{H}$-*graph* are formed by the common nodes between the paths.

## 3 Problem Definition

In this section, we formally define the problem of evaluating reachability queries over a path collection. We introduce the basic aspects of the problem and our notation for the rest of the paper. We begin by defining the notion of a path collection over a set of nodes.

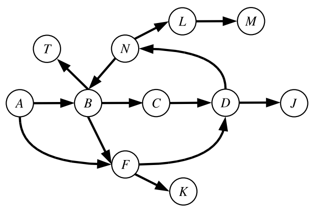
**Definition 1 (Path).** *Let $V$ be a set of nodes. A* path $p(v_1, \ldots, v_k)$ *over $V$ is a sequence of distinct nodes $(v_1, \ldots, v_k) \in V$. By $nodes(p)$ we denote the set of nodes in $p$. The length of a path $p$, denoted by $l_p$, is the number of contained nodes, i.e., $l_p = |nodes(p)|$.*

**Definition 2 (Path collection).** *Let $V$ be a set of nodes. A path collection over $V$, denoted by $\mathbf{P}$, is a set of paths $\{p_1, \ldots, p_m\}$ over $V$. By $nodes(\mathbf{P})$ we denote the set of nodes in $\mathbf{P}$.*

*Example 1.* Figure 1(a) illustrates an example of a path collection $\mathbf{P} = \{p_1, p_2, p_3, p_4, p_5\}$ over $V = \{A, ..., Z\}$.

$p_1$ $(A, B, C, D, J)$
$p_2$ $(A, F, D, N, B, T)$
$p_3$ $(N, L, M)$
$p_4$ $(D, N, B, F, K)$
$p_5$ $(A, F, K)$

(a)



(b)

**Fig. 1.** (a) A path collection $\mathbf{P}$, (b) the underlying graph $G_{\mathbf{P}}$ of $\mathbf{P}$

Next, we define the family of reachability queries over a path collection.

**Definition 3 (Reachability queries).** *Let $\mathbf{P}$ be a path collection, and $v_s$, $v_t$ be two nodes in $nodes(\mathbf{P})$. The family of reachability queries deals with the following problems:*

- *Determine whether there exists a path from $v_s$ to $v_t$. This query is denoted by* $\texttt{reach}(v_s, v_t)$.

– *Identify a path from $v_s$ to $v_t$. This query is denoted by* `path`$(v_s, v_t)$.

In this paper, we deal with the problem of evaluating reachability queries over path collections. To evaluate reachability queries we exploit only the transitions between the nodes contained in the paths of the collection. The collections can be frequently updated with new paths that may involve a number of new nodes. Therefore, we also have to efficiently deal with massive updates.

Given a path collection $\mathbf{P}$, one can construct a graph that contains all the reachability information present in $\mathbf{P}$.

**Definition 4 (Underlying graph).** *Let $\mathbf{P}$ be a path collection. The underlying graph of $\mathbf{P}$, denoted by $G_{\mathbf{P}}(V,E)$ or simply $G_{\mathbf{P}}$, is a directed graph that contains all the nodes, $V = nodes(\mathbf{P})$, and all the direct transitions of $\mathbf{P}$, $E = \{\ (u,v) : (\ldots,u,v,\ldots) \in \mathbf{P}\}$.*

It is easy to verify that a path collection $\mathbf{P}$ over set $V$ and the underlying graph $G_{\mathbf{P}}(V,E)$ are equivalent with respect to reachability queries. For example, one can answer `path`$(F, C)$ over the path collection in Figure 1(a) exploiting $G_{\mathbf{P}}$ graph in Figure 1(b). Therefore, a simple solution to the problem is a search algorithm that exploits the adjacency lists of the graph, with the additional benefits that it imposes minimum construction cost and deals easily with massive updates. In the rest of this work, we consider paths to be first class citizens and propose alternative search-based methods for the task at hand.

## 4 Evaluating Reachability Queries over Path Collections

Section 4.1 introduces the *path-first search* algorithm, termed pfs, for evaluating reachability queries over path collections and Section 4.2 discusses optimizations based on the $\mathcal{P}$-*Index* structure.

### 4.1 The Path-First Search Algorithm

The basic idea of the pfs, illustrated in Figure 2, is to examine entire paths at once rather than single nodes. The algorithm takes the collection $\mathbf{P}$, the source $v_s$ and target node $v_t$ as inputs and returns a path connecting them, if one exists, or null, otherwise. The algorithm employs the following data structures: (i) the search stack $\mathcal{Q}$, (ii) the history set $\mathcal{H}$, which contains all nodes that have been pushed in $\mathcal{Q}$, and (iii) the ancestor set $\mathcal{A}$, which stores the direct ancestor of each node in $\mathcal{Q}$. $\mathcal{H}$ is used to avoid cycles and $\mathcal{A}$ to extract answer paths. Note that pfs visits each node once and, thus, there is a single entry per node in $\mathcal{A}$.

The pfs algorithm proceeds similarly to depth-first search as follows. Initially, the stack $\mathcal{Q}$ and $\mathcal{H}$ contain source node $v_s$ (Lines 1–2). Further, the entry $\langle v_s, \varnothing\rangle$ is inserted in $\mathcal{A}$ (Line 3) denoting that $v_s$ is the source node. The algorithm proceeds examining the contents of the stack (Lines 4–16). The current top node $v_n$ is popped from $\mathcal{Q}$ (Line 5) and checked against target $v_t$. If they are equal the search terminates and the path is extracted by the ConstructPath method (Line

```
Algorithm pfs
Input: nodes $v_s$ and $v_t$ of a path collection $\mathbf{P}$
Output: a path from $v_s$ to $v_t$
Parameters:
 stack $\mathcal{Q}$:    // the search stack
 set $\mathcal{H}$:      // contains all nodes pushed in $\mathcal{Q}$
 set $\mathcal{A}$:      // contains the direct ancestor of each node in $\mathcal{H}$
Method:
  1. push($v_s$, $\mathcal{Q}$);
  2. insert $v_s$ in $\mathcal{H}$
  3. insert $\langle v_s, \varnothing \rangle$ in $\mathcal{A}$;
  4. while $\mathcal{Q}$ is not empty do
  5.    let $v_n$ = pop($\mathcal{Q}$);
  6.    if $v_n$ is equal to $v_t$ then return ConstructPath($v_s, v_n, \mathcal{A}$);
  7.    for each path $p \in \mathbf{P}$ containing $v_n$ do
  8.       let $v_p$ be the node after $v_n$ in $p$;
  9.       while $v_p \notin \mathcal{H}$ do   // access each node $v_p$ after $v_n$ in $p$ until the first $v_p$ contained in $\mathcal{H}$
 10.          push($v_p$, $\mathcal{Q}$);
 11.          insert $v_p$ in $\mathcal{H}$;
 12.          insert $\langle v_p, v_p^- \rangle$ in $\mathcal{A}$, where $v_p^-$ is the direct ancestor of $v_p$ in $p$;
 13.          let $v_p$ be the next node in $p$;
 14.       end while
 15.    end for
 16. end while
 17. return null;
```

**Fig. 2.** Algorithm pfs

6). Specifically, starting from $v_t$, ConstructPath uses the ancestor information of $\mathcal{A}$ to backtrack to source $v_s$.

If the target is not found, pfs considers all paths that contain $v_n$ and examines their contents (Lines 7–15). Fix such a path $p$ and let $v_p$ denote the node that follows current top node $v_n$ in $p$ (Line 8). Next, a while loop begins checking if $v_p$ has never been pushed in $\mathcal{Q}$ (i.e., $v_p \notin \mathcal{H}$). If the check succeeds, $v_p$ is pushed in $\mathcal{Q}$ and inserted in $\mathcal{H}$ (Lines 10–11). In addition, the entry $\langle v_p, v_p^- \rangle$, where $v_p^-$ is the direct ancestor of $v_p$ in path $p$, is inserted in $\mathcal{A}$ (Line 12). Last, $v_p$ is updated to the next node in $p$ (Line 13) and the while loop continues checking the new $v_p$. The condition on Line 9 ensures that only nodes that have not been previously enqueued are inserted in $\mathcal{Q}$; hence, pfs avoids cycles.

*Example 2.* We illustrate the pfs algorithm for the query `path`$(F, C)$ on the path collection of Figure 1(a). Initially, we have (Lines 1–3):

$$\mathcal{Q} = \{F\},\ \mathcal{H} = \{F\} \text{ and } \mathcal{A} = \{\langle F, \varnothing \rangle\}.$$

At the first iteration of the outer while loop, the algorithm pops $F$ from $\mathcal{Q}$ and identifies paths $p_2$, $p_4$ and $p_5$ that contain $F$.

– When processing $p_2(A, F, D, N, B, T)$, the algorithm adds to $\mathcal{Q}$ and $\mathcal{H}$, nodes $D$, $N$, $B$ and $T$, and to $\mathcal{A}$ pairs $\langle D, F \rangle$, $\langle N, D \rangle$, $\langle B, N \rangle$ and $\langle T, B \rangle$.
– When processing $p_4(D, N, B, F, K)$, the algorithm adds to $\mathcal{Q}$ and $\mathcal{H}$, node $K$, and to $\mathcal{A}$ pair $\langle K, F \rangle$.
– When processing $p_5(A, F, K)$, the algorithm does not add anything to $\mathcal{Q}$, $\mathcal{H}$ and $\mathcal{A}$ since there are no new nodes after the current node $F$ ($K$ has been enqueued).

After the first iteration, we have:

$$\mathcal{Q} = \{D, N, B, T, K\},$$
$$\mathcal{H} = \{F, D, N, B, T, K\} \text{ and}$$
$$\mathcal{A} = \{\langle F, \varnothing\rangle, \langle D, F\rangle, \langle N, D\rangle, \langle B, N\rangle, \langle T, B\rangle, \langle K, F\rangle\}.$$

The algorithm proceeds in a similar manner. After the fourth iteration, we have:

$$\mathcal{Q} = \{D, N, C\},$$
$$\mathcal{H} = \{F, D, N, B, T, K, C\} \text{ and}$$
$$\mathcal{A} = \{\langle F, \varnothing\rangle, \langle D, F\rangle, \langle N, D\rangle, \langle B, N\rangle, \langle T, B\rangle, \langle K, F\rangle \; \langle C, B\rangle\}.$$

At the fifth iteration, pfs pops $C$ (the target) from stack $\mathcal{Q}$ and terminates the search. ConstructPath returns answer path $(F, D, N, B, C)$ by scanning $\mathcal{A}$.

Algorithm pfs terminates the search when the target node is popped out of stack $\mathcal{Q}$. An alternative approach is to check whether both current search node and the target node are contained in a path of the collection and terminate search without visiting any other node. In the next section, we discuss this improvement and present an extension to pfs called pfsP.

## 4.2 $\mathcal{P}$-*Index*: Indexing Path Collections

In this section, we describe the *path collection index* $\mathcal{P}$-*Index*, an inverted index on the path collection. We can take advantage of $\mathcal{P}$-*Index* in two ways: (i) for accessing all paths that contain current search node (Line 7 in Figure 2), and (ii) for enforcing a quick termination condition.

**Definition 5 ($\mathcal{P}$-*Index*).** *The* path collection index *of* $\mathbf{P}$*, denoted as* $\mathcal{P}$-*Index* ($\mathbf{P}$)*, consists of* paths *lists for all nodes in* $\mathbf{P}$*. The list* $paths[v_i]$ *for node* $v_i$ *contains entries* $\langle p_j{:}o_{ij}\rangle$*, where* $o_{ij}$ *indicates the position of* $v_i$ *in* $p_j$*, for all paths* $p_j$ *that include* $v_i$*. The entries are stored sorted by their path identifier* $p_j$*.*

*Example 3.* Table 1 illustrates the path collection index $\mathcal{P}$-*Index*($\mathbf{P}$) for the collection $\mathbf{P}$ presented in Figure 1(a).

**Table 1.** $\mathcal{P}$-*Index* for the path collection $\mathbf{P}$ in Figure 1(a)

| **node** | *paths* **list** |
|---|---|
| $A$ | $\langle p_1{:}1\rangle, \langle p_2{:}1\rangle, \langle p_5{:}1\rangle$ |
| $B$ | $\langle p_1{:}2\rangle, \langle p_2{:}5\rangle, \langle p_4{:}3\rangle$ |
| $C$ | $\langle p_1{:}3\rangle$ |
| $D$ | $\langle p_1{:}4\rangle, \langle p_2{:}3\rangle, \langle p_4{:}1\rangle$ |
| $F$ | $\langle p_2{:}2\rangle, \langle p_4{:}4\rangle, \langle p_5{:}2\rangle$ |
| $J$ | $\langle p_1{:}5\rangle$ |
| $K$ | $\langle p_4{:}5\rangle, \langle p_5{:}3\rangle$ |
| $L$ | $\langle p_3{:}2\rangle$ |
| $M$ | $\langle p_3{:}3\rangle$ |
| $N$ | $\langle p_2{:}4\rangle, \langle p_3{:}1\rangle, \langle p_4{:}2\rangle$ |
| $T$ | $\langle p_2{:}6\rangle$ |

We introduce pfsP as an extension to pfs algorithm that exploits $\mathcal{P}$-*Index*. Algorithm pfsP identifies all paths that contain a node $v_n$ by performing a linear scan of list $paths[v_n]$.

Furthermore, pfsP exploits $\mathcal{P}$-*Index* to define a fast termination condition. Assume that node $v_n$ has just been popped out (Line 5). The search can be terminated if there exists a path $p_c$ in the collection that contains both $v_n$ and target $v_t$, such that, $v_t$ comes after $v_n$. Specifically, pfsP looks for entries $\langle p_c{:}o_{nc}\rangle$, $\langle p_c{:}o_{tc}\rangle$ in lists $paths[v_n]$, $paths[v_t]$ respectively, such that $o_{nc} < o_{tc}$. The procedure is similar to a merge-join that finishes as soon as such a path is found or one of the lists is traversed to the end.

The pfsP is similar to pfs with the exception that it performs the described check. The improved termination condition can be included in Figure 2 by changing Line 6 to:

```
6. if there is a path $p_c \in \mathbf{P}$ containing $v_n$ before $v_t$ then
     return ConstructPathP($v_s, v_n, v_t, \mathcal{A}, p_c$);
```

To construct path$(v_s, v_n)$, ConstructPathP method first calls ConstructPath $(v_s, v_n, \mathcal{A})$. Then, it concatenates path$(v_s, v_n)$ with the part of $p_c$ from $v_n$ up to $v_t$. During concatenation the method ensures that each node is contained only once in the answer path. For example, consider `path`$(A, T)$. After joining $paths[D] = \{\langle p_1{:}4\rangle, \langle p_2{:}3\rangle, \langle p_4{:}1\rangle\}$ and $paths[T] = \{\langle p_2{:}6\rangle\}$ lists we identify common path $p_2$. The ConstructPathP method first constructs path$(A, D) = (A, B, C, D)$ using set $\mathcal{A}$ and then concatenates it with the part of $p_1(A, F, D, N, B, T)$ from $D$ up to $T$. Since node $B$ is contained in path$(A, D)$ the answer path is $(A, B, T)$.

*Example 4.* We illustrate the pfsP algorithm for the query `path`$(F, C)$ on the path collection $\mathbf{P}$ of Example 2 exploiting the join procedure of the *paths* lists. We use $\mathcal{P}$-*Index*$(\mathbf{P})$ presented in Table 1.

The first three iterations are identical to the first three iterations of the pfs algorithm presented in Example 2. Summarizing, after these iterations the stack and the sets of pfsP are as follows.

$$\mathcal{Q} = \{D, N, B\},$$
$$\mathcal{H} = \{F, D, N, B, T, K\} \text{ and}$$
$$\mathcal{A} = \{\langle F, \varnothing\rangle, \langle D, F\rangle, \langle N, D\rangle, \langle B, N\rangle, \langle T, B\rangle, \langle K, F\rangle\}.$$

At the fourth iteration of the outer while loop, pfsP pops $B$. To execute Line 6, we join the *paths* list of current search node $B$, $paths[B] = \{\langle p_1{:}2\rangle, \langle p_2{:}\ 5\rangle, \langle p_4{:}\ 3\rangle\}$ with the *paths* list for target $C$, $paths[C] = \{\langle p_1{:}3\rangle\}$. The join procedure identifies entries $\langle p_1{:}2\rangle$, $\langle p_1{:}3\rangle$ for common path $p_c = p_1$. Since in $p_1$, $B$ is before $C$, the search terminates successfully. The answer path $(F, D, N, B, C)$ is the concatenation of $(F, D, N, B)$ (which corresponds to the path from source $F$ to current node $B$ and is constructed using set $\mathcal{A}$) and $(B, C)$ (which corresponds to the part of $p_1$ that connect $B$ to target $C$).

## 5 Capturing Reachability Information Using $\mathcal{H}$-*graphs*

Section 5.1 introduces the $\mathcal{H}$-*graph* and its associated structure $\mathcal{H}$-*Index*. Section 5.2 discusses the extension of pfs using the $\mathcal{H}$-*Index*.

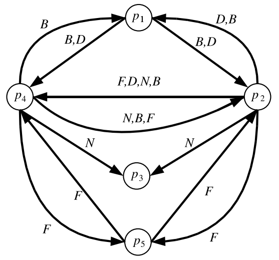
## 5.1 The $\mathcal{H}$-*graph* and Its $\mathcal{H}$-*Index*

The $\mathcal{H}$-*graph* provides additional reachability information by identifying shared nodes and, thus, possible transitions, among paths.

**Definition 6** **($\mathcal{H}$-*graph*).** *Let* $\mathbf{P} = \{p_1, ..., p_n\}$ *be a path collection. The* $\mathcal{H}$-*graph of* $\mathbf{P}$*, denoted by* $\mathcal{H}$-*graph*$(\mathbf{P})$*, is a labelled directed graph* $(V, E)$ *such that* $V$ *consists of all paths in* $\mathbf{P}$ *and a labelled edge* $(p_i, p_j, v) \in E$ *if paths* $p_i$*,* $p_j$ *have a common node* $v \in nodes(\mathbf{P})$*, termed* link*, which is neither the first node of* $p_i$ *nor the last of* $p_j$*.*

Given a path collection $\mathbf{P}$ and $\mathcal{P}$-*Index*$(\mathbf{P})$, $\mathcal{H}$-*graph*$(\mathbf{P})$ is constructed as follows. For each node $v_k \in nodes(\mathbf{P})$ and each pair of entries $\langle p_i{:}o_{ki}\rangle, \langle p_j{:}o_{kj}\rangle \in paths[v_k]$, we construct a directed edge from $p_i$ to $p_j$ in $\mathcal{H}$-*graph*$(\mathbf{P})$ and label it with link $v_k$. Intuitively, edge $(p_i, p_j, v)$ denotes that all nodes in $p_i$ before link $v_k$ can reach the nodes after $v_k$ in $p_j$. If the link lies in the beginning of $p_i$ or at the end of $p_j$, there is no useful reachability information since no node is contained before $v_k$ in $p_i$ or after $v_k$ in $p_j$, and hence the edge is omitted from $\mathcal{H}$-*graph*.

*Example 5.* Figure 3(a) illustrates $\mathcal{H}$-*graph*$(\mathbf{P})$ for the path collection $\mathbf{P}$ of Figure 1(a). To increase readability, multiple edges between the same pair of paths are collapsed into a single edge with multiple labels. For example, the single edge from $p_4$ to $p_2$ labelled with $N, B, F$ links corresponds to edges $(p_4, p_2, N)$, $(p_4, p_2, B)$ and $(p_4, p_2, F)$. Note that edge $(p_4, p_1, D)$ is not included since $D$ is the first node in $p_4$.



(a)

| path | *edges* list |
|---|---|
| $p_1$ | $\langle p_2, B{:}2{:}5\rangle, \langle p_2, D{:}4{:}3\rangle, \langle p_4, B{:}2{:}3\rangle, \langle p_4, D{:}4{:}1\rangle$ |
| $p_2$ | $\langle p_1, D{:}3{:}4\rangle, \langle p_1, B{:}5{:}2\rangle, \langle p_3, N{:}4{:}1\rangle, \langle p_4, F{:}2{:}4\rangle, \langle p_4, D{:}3{:}1\rangle, \langle p_4, N{:}4{:}2\rangle, \langle p_4, B{:}5{:}3\rangle, \langle p_5, F{:}2{:}2\rangle$ |
| $p_3$ | |
| $p_4$ | $\langle p_1, B{:}3{:}2\rangle, \langle p_2, N{:}2{:}4\rangle, \langle p_2, B{:}3{:}5\rangle, \langle p_2, F{:}4{:}2\rangle, \langle p_3, N{:}2{:}1\rangle, \langle p_5, F{:}4{:}2\rangle$ |
| $p_5$ | $\langle p_2, F{:}2{:}2\rangle, \langle p_4, F{:}2{:}4\rangle$ |

(b)

**Fig. 3.** (a) $\mathcal{H}$-*graph* $(\mathbf{P})$ of the path collection $\mathbf{P}$ in Figure 1(a), (b) $\mathcal{H}$-*Index* for $\mathcal{H}$-*graph* $(\mathbf{P})$

The $\mathcal{H}$-*graph* of a path collection $\mathbf{P}$ is stored in a modified adjacency list representation denoted as $\mathcal{H}$-*Index*.

**Definition 7** **($\mathcal{H}$-*Index*).** *The* $\mathcal{H}$-*graph* index *of* $\mathbf{P}$*, denoted as* $\mathcal{H}$-*Index*$(\mathbf{P})$*, consists of* edges *lists for all paths in* $\mathbf{P}$*. The list* $edges[p_i]$ *for path* $p_i$ *has entries of the form* $\langle p_j, v_k : o_{ki} : o_{kj}\rangle$*, for each* $(p_i, p_j, v_k)$ *edge of* $\mathcal{H}$-*graph*$(\mathbf{P})$*, where* $o_{ki}$ *(*$o_{kj}$*) denotes the position of the link* $v_k$ *in path* $p_i$ *(*$p_j$*). Entries are sorted primarily by the path* $p_j$ *of the outgoing edge, and secondarily by* $o_{ki}$*.*

*Example 6.* Figure 3(b) illustrates the $\mathcal{H}$-*Index*($\mathbf{P}$) of the $\mathcal{H}$-*graph*($\mathbf{P}$) presented in Figure 3(a).

## 5.2 The pfsH Algorithm

The $\mathcal{H}$-*graph* captures intersections among paths, and hence contains additional information about nodes' reachability compared to that included in the paths alone. To illustrate this, consider node $F$ of path $p_2$ and node $C$ of path $p_1$ of the collection in Figure 1(a). The information in $\mathcal{H}$-*Index* suffices to show that a path from $F$ to $C$ exists. In particular, the entry $\langle p_1, B{:}5{:}2\rangle$ of $edges[p_2]$ in $\mathcal{H}$-*Index* denotes that there is way from $p_2$ to $p_1$ via $B$. Further, from $\mathcal{P}$-*Index* one derives that $B$ is after $F$ in $p_2$ and before $C$ in $p_1$. Hence a path $(F, D, N, B, C)$ can be constructed by combining paths $p_2$ and $p_1$.

The above observation is the main idea of pfsH algorithm. Consider the query $\texttt{path}(v_s, v_t)$ and assume that current search node is $v_n$. For each path $p_i$ that contains $v_n$, the algorithm checks whether an edge $(p_i, p_j, v_k)$ in $\mathcal{H}$-*graph* satisfies three conditions: (i) $p_j$ contains the target node $v_t$, (ii) link $v_k$ is after current search node $v_n$ in $p_i$, and (iii) $v_k$ is before $v_t$ in $p_j$. If these hold, a path from $v_n$ to target $v_t$, via $v_k$ exists, and thus a path from source $v_s$ to $v_t$ can be found.

Algorithm pfsH is similar to pfs with the exception that it introduces two termination conditions. First, before initializing stack $\mathcal{Q}$ and sets $\mathcal{H}$, $\mathcal{A}$ in Figure 2 (Lines 1-3), the algorithm checks whether there exists a path $p_c$ in the collection containing source $v_s$ before target $v_t$. To perform this check pfsH exploits the join procedure of $paths[v_s]$ and $paths[v_t]$ lists introduced for pfsP in Section 4.2. If a path $p_c$ is identified the search terminates and the ConstructPathP method returns the part of $p_c$ from $v_s$ to $v_t$ as the answer path.

Otherwise, as soon as a new path $p_i$ containing current search node $v_n$ in position $o_{ni}$ is examined (after Line 7 in Figure 2), pfsH checks whether there exists an edge $(p_i, p_j, v_k)$ in $\mathcal{H}$-*graph* satisfying the aforementioned three conditions. The algorithm scans lists $edges[p_i]$ and $paths[v_t]$ from $\mathcal{H}$-*Index*($\mathbf{P}$) and $\mathcal{P}$-*Index*($\mathbf{P}$), respectively, similar to a merge-join as both are sorted on the path identifier. The scan terminates when $\langle p_j, v_k{:}o_{ki}{:}o_{kj}\rangle$ in $edges[p_i]$ and $\langle p_j{:}o_{tj}\rangle$ in $paths[v_t]$ match, i.e., correspond to the same path $p_j$ (condition (i)), and additionally $o_{ki} > o_{ni}$ (condition (ii)) and $o_{kj} < o_{tj}$ (condition (iii)).

When a qualifying entry $\langle p_j, v_k : o_{ki} : o_{kj}\rangle$ in $edges[p_i]$ is found, pfsH first constructs path$(v_s, v_n)$, calling ConstructPath$(v_s, v_n, \mathcal{A})$, and then concatenates it with the part of $p_i$ from $v_n$ to $v_k$ and the part of $p_j$ from $v_k$ to $v_t$.

For a more detailed presentation of pfsH algorithm see [11].

*Example 7.* We illustrate the pfsH algorithm for the query $\texttt{path}(F, C)$ on the path collection of $\mathbf{P}$ of Example 2. Algorithm pfsH exploits $\mathcal{H}$-*Index*($\mathbf{P}$) presented in Figure 3(b) and $\mathcal{P}$-*Index*($\mathbf{P}$) of Table 1.

First, we check whether exists a path in $\mathbf{P}$ containing source $F$ before target $C$. The join between $paths[F] = \{\langle p_2{:}2\rangle, \langle p_4{:}4\rangle, \langle p_5{:}2\rangle\}$ and $paths[C] = \{\langle p_1{:}3\rangle\}$ lists results in no common path. Thus, we need to further search the collection.

At the first iteration of the outer while loop, pfsH pops $F$. Node $F$ is contained in paths $p_2(A, F, D, N, B, T)$, $p_4(D, N, B, F, K)$ and $p_5(A, F, K)$. Then, we check the termination condition for paths $p_2$, $p_4$ and $p_5$ and perform a join of the corresponding *edges* list with $paths[C] = \{\langle p_1 : 3\rangle\}$. The join of $edges[p_2] = \{\langle p_1, D{:}3{:}4\rangle, \langle p_1, B{:}5{:}2\rangle, \langle p_3, N{:}4{:}1\rangle, \langle p_4, F{:}2{:}5\rangle, \langle p_4, D{:}3{:}1\rangle, \langle p_4, N{:}4{:}2\rangle, \langle p_4, B{:}5{:}3\rangle, \langle p_5, F{:}2{:}2\rangle\}$ with $paths[C] = \{\langle p_1{:}3\rangle\}$ results in common path $p_1$ (condition (i)) with the link $B$ of $(p_2, p_1, B)$ edge contained after $F$ in $p_2$ (condition (ii)) and before $C$ in $p_1$ (condition (iii)). Thus, the answer path is $(F, D, N, B, C)$.

## 6 Updating Path Collections

Updating a path collection involves adding new paths. To include a new path $p_j$ in a collection, we need (a) to insert the entry $\langle p_j{:}o_{ij}\rangle$ in $paths[v_i]$ for each node $v_i$ contained in $p_j$ (update $\mathcal{P}$-*Index*), and (b) to update $edges[p_j]$ and the *edges* lists of the paths containing each node in $p_j$ (update $\mathcal{H}$-*Index*).

In practice, path collections are usually very large to fit in main memory. Therefore, both $\mathcal{P}$-*Index* and $\mathcal{H}$-*Index* of a collection are stored as inverted files on secondary storage and maintained mainly by batch, offline updates. In other words, we usually update the collection with a set of new paths. A requirement for the inverted files to work efficiently is to store the inverted lists, like *paths* and *edges* lists, in a contiguous way on secondary storage. Due to this requirement the naïve solution to deal with each new path separately is not efficient for updating the collection. A common approach to this problem is to built a $\mathcal{P}$-*Index* and an $\mathcal{H}$-*Index* considering the new paths as inverted indices in main memory and to exploit them for evaluating the queries in parallel with the disk-based $\mathcal{P}$-*Index* and $\mathcal{H}$-*Index* of the collection.

Each time a set of new paths arrives, we update only the memory-based indices with minimum cost. Then, to update the disk-based indices, there are three possible strategies ([12]): (a) rebuilding them from scratch using both the old and the new paths, (b) merging them with the memory resident ones and (c) lazily updating the *paths* and the *edges* lists when they are retrieved from

```
Procedure updateMP
Inputs: memory-based $\mathcal{P}$-Index(P) $MP$, set of new paths $\mathcal{U}$
Output: updated memory-based $\mathcal{P}$-Index(P) $MP$
Method:
1. for each new path $p_j$ in $\mathcal{U}$ do
2.    for each node $v_k$ in $p$ do
3.       append $\langle p_j{:}o_{kj}\rangle$ entry at the end of $paths[v_k]$ in $MP$;
4.    end for
5. end for
Procedure mergeP
Inputs: updated memory-based $\mathcal{P}$-Index(P) $MP$, disk-based $\mathcal{P}$-Index(P) $DP$
Output: updated disk-based $\mathcal{P}$-Index(P) $DP$
Method:
1. for each node $v$ in $MP$ do
2.    append contents of $paths[v]$ of $MP$ at the end of $paths[v]$ of $DP$;
3.    write new $paths[v]$ on $DP$;
4. end for
```

**Fig. 4.** Procedures for updating $\mathcal{P}$-*Index*

disk during query evaluation. In our work, we adopt the second strategy for updating the disk-based indices.

Figures 4 and 5 illustrate the procedures for updating the memory-based indices with the new paths (Procedures updateMP and updateMH) and the merging procedures of the disk-based indices with the memory-based ones (Procedures mergeP and mergeH). Procedures updateMP and updateMH work similarly with the procedures for creating disk-based $\mathcal{P}$-*Index* and $\mathcal{H}$-*Index* respectively, from scratch. Especially for updateMH, we also need to create entries considering both the new paths and the existing paths of the collection (Lines 6-9). Finally, Procedures mergeP and mergeH merge disk-based *paths* and *edges* lists respectively with the memory resident ones, and then write the new lists on disk.

```
Procedure updateMH
Inputs: updated memory-based $\mathcal{P}$-Index($\mathbf{P}$) $MP$, disk-based $\mathcal{P}$-Index($\mathbf{P}$) $DP$, memory-based $\mathcal{H}$-Index($\mathbf{P}$) $MH$
Output: updated memory-based $\mathcal{H}$-Index($\mathbf{P}$) $MH$
Method:
 1. for each node $v_k$ in $MP$ do
 2.    for each pair of entries $\langle p_i{:}o_{ki}\rangle, \langle p_j{:}o_{kj}\rangle$ in $paths[v_k]$ of $MP$ do
 3.       if $o_{ki} > 1$ and $o_{kj} < l_{p_j}$ then insert $\langle p_j, v_k{:}o_{ki}{:}o_{kj}\rangle$ in $edges[p_i]$ of $MH$;
 4.       if $o_{kj} > 1$ and $o_{ki} < l_{p_i}$ then insert $\langle p_i, v_k{:}o_{kj}{:}o_{ki}\rangle$ in $edges[p_j]$ of $MH$;
 5.    end for
 6.    for each pair of entries $\langle p_i{:}o_{ki}\rangle, \langle p_j{:}\ o_{kj}\rangle$ where $p_i \in paths[v_k]$ of $MP$ and $p_j \in paths[v_k]$ of
         $DP$ do
 7.       if $o_{ki} > 1$ and $o_{kj} < l_{p_j}$ then insert $\langle p_j, v_k{:}o_{ki}{:}o_{kj}\rangle$ in $edges[p_i]$ of $MH$;
 8.       if $o_{kj} > 1$ and $o_{ki} < l_{p_i}$ then insert $\langle p_i, v_k{:}o_{kj}{:}o_{ki}\rangle$ in $edges[p_j]$ of $MH$;
 9.    end for
10. end for
```

```
Procedure mergeH
Inputs: updated memory-based $\mathcal{H}$-Index($\mathbf{P}$) $MH$, disk-based $\mathcal{H}$-Index($\mathbf{P}$) $DH$
Output: updated disk-based $\mathcal{H}$-Index($\mathbf{P}$) $DH$
Method:
 1. for each path $p$ in $MH$ do
 2.    merge $edges[p]$ of $DH$ with $edges[p]$ of $MH$;
 3.    write new $edges[p]$ on $DH$;
 4. end for
```

**Fig. 5.** Procedures for updating $\mathcal{H}$-*Index*

## 7 Experiments

We present an experimental evaluation of our methods demonstrating their efficiency. We compare the pfsP and pfsH algorithms against conventional depth-first search which operates on the underlying graph $G_{\mathbf{P}}$, indexed by adjacency lists. All indices, i.e., $\mathcal{P}$-*Index* and $\mathcal{H}$-*Index* for the collections, and the adjacency lists for $G_{\mathbf{P}}$, are implemented as inverted files using the Berkeley DB storage engine. All algorithms are implemented in C++ and compiled with gcc. The experimental evaluation was performed on a 3 Ghz Intel Core 2 Duo CPU.

For updating the adjacency lists of $G_{\mathbf{P}}$ graph, we adopt an approach similar to the one for $\mathcal{P}$-*Index* and $\mathcal{H}$-*Index*. In addition, we choose not to check whether the new paths contain a transition between two nodes more than once or if a transition is already included as an edge in $G_{\mathbf{P}}$ graph. Instead, duplicates are removed while merging the disk-based adjacency lists with the memory-based ones. This approach allows for fast updates on the adjacency lists and $G_{\mathbf{P}}$ graph, at the expense of increased main memory utilization.

We generate synthetic path collections to test the methods. We identify five experimental parameters: (a) $|\mathbf{P}|$: the number of paths in the collection, (b) $l_{avr}$: the average path length, (c) $|V|$: the number of distinct nodes in the paths, (d) $zipf$: the order of Zipfian distribution of node frequency, and (e) $U$: the update factor. The path collections contain 50000 up to 500000 paths. The average length of each path varies between 5 to 30 nodes. Path collections include 10000 up to 500000 distinct nodes. Note that varying the number of nodes in the collection also affects the number of link (common) nodes and the possible transitions between the paths. Node frequency is a moderately skewed Zipfian distribution of order $zipf$ that varies from 0 up to 0.8. Note that nodes with high frequency are contained in a lot of paths. An update factor $U$ means that there are $U\% \cdot |\mathbf{P}|$ new paths to be added to the collection $\mathbf{P}$. Table 2 summarizes all parameters.

We perform four sets of experiments to show the effects on the size and the construction time of the indices, as well as on the performance of the algorithms for evaluating 5000 random reachability queries. In each set, we vary one of $|\mathbf{P}|$, $l_{avg}$, $V$, $zipf$ while we keep the remaining three parameters fixed to their default values (see Table 2). In the fifth set of experiments, we study the updates of the path collections. We vary only the $U$ parameter while we set the remaining four fixed to their default values.

**Table 2.** Experimental parameters

| parameter | values | default value |
|---|---|---|
| $\|\mathbf{P}\|$ | 50000, 100000, 500000 | 100000 |
| $l_{avg}$ | 5, 10, 30 | 10 |
| $\|V\|$ | 10000, 50000, 100000, 500000 | 100000 |
| $zipf$ | 0, 0.3, 0.6, 0.8 | 0.6 |
| $U$ | 1%, 5%, 10% | - |

**Varying the number of paths in the collection.** Figure 6(a) illustrates the effect on the index size. We note that in all cases, $\mathcal{H}$-*Index* requires at least one order of magnitude more space than the other two indices. $\mathcal{P}$-*Index* is slightly larger than the adjacency lists. As $|\mathbf{P}|$ increases all indices require more disk space. The size of the adjacency lists increases, because the path collections include more direct transitions between path nodes resulting in more dense $G_{\mathbf{P}}$ graphs. As expected $\mathcal{P}$-*Index* requires more space since each node is contained in more paths and therefore, the length of the *paths* lists increases. Finally, as $|\mathbf{P}|$ increases, the paths have more nodes in common, which means that the length of the *edges* lists increases too. Thus, $\mathcal{H}$-*Index* also requires more disk space.

Figure 6(b) shows the effect on the construction time of the indices. As $|\mathbf{P}|$ increases the construction of all indices takes more time. We notice that the creation time of the adjacency lists is almost one order of magnitude higher than the time for $\mathcal{P}$-*Index*, in all cases. This is due to the fact that, we first need to construct $G_{\mathbf{P}}$ graph by removing repeated transitions between nodes.

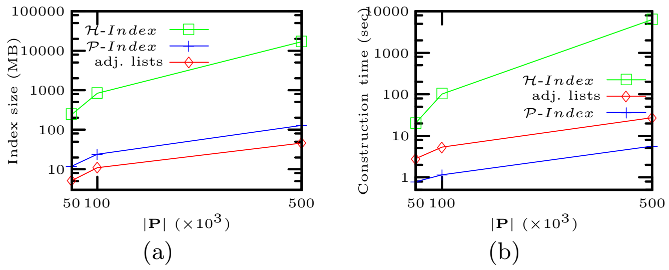


**Fig. 6.** (a) Index size, and (b) construction time varying $|\mathbf{P}|$, for $l_{avg} = 10$, $|V| = 100000$, $zipf = 0.6$

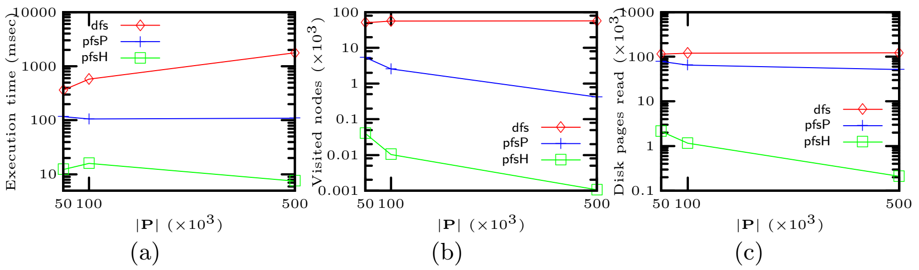


**Fig. 7.** (a) Average execution time, (b) average number of visited nodes, (c) average number of disk pages read varying $|\mathbf{P}|$, for $l_{avg} = 10$, $|V| = 100000$, $zipf = 0.6$

On the other hand, the construction time of *𝓗-Index* is always approximately one order of magnitude higher than the time of the other indices.

Figure 7(a) presents the effects of $|\mathbf{P}|$ on the query execution time. In all cases, the average execution time of pfsP and pfsH is lower than that of dfs. pfsP is always almost one order of magnitude faster than dfs. pfsH is two orders of magnitude faster than dfs. As $|\mathbf{P}|$ increases, the execution time of dfs increases too. This is expected since $G_{\mathbf{P}}$ graph becomes more dense. In contrast, pfsP is less affected by the increase of $|\mathbf{P}|$, whereas the execution time of pfsH decreases. This is because, the length of the *paths* lists in *𝓟-Index* and the *edges* lists in *𝓗-Index* increases and it is very likely that the join procedures in pfsP and pfsH will identify common paths. Thus pfsP and pfsH, in all cases, need to visit fewer nodes to answer a query as Figure 7(b) shows. Figure 7(c) confirms the above observations with respect to the number of I/Os.

**Varying the average length of the paths in the collection.** Figure 8(a) shows the effects of $l_{avg}$ on the disk space required to store indices. Similarly to the case of increasing $|\mathbf{P}|$, we notice that the size of *𝓗-Index* is more than one order of magnitude larger compared to the size of the adjacency lists of $G_{\mathbf{P}}$ graph. In contrast, *𝓟-Index* is slightly larger than the adjacency lists. As $l_{avg}$ increases, there are more direct transitions between path nodes. Thus, $G_{\mathbf{P}}$ graph becomes more dense and its adjacency lists contain more nodes. Finally, as (a) the $l_{avg}$ increases, and (b) $|V|$ remains fixed, the number of occurrences in the paths for each node increases. This results to longer *paths* lists in *𝓟-Index* and

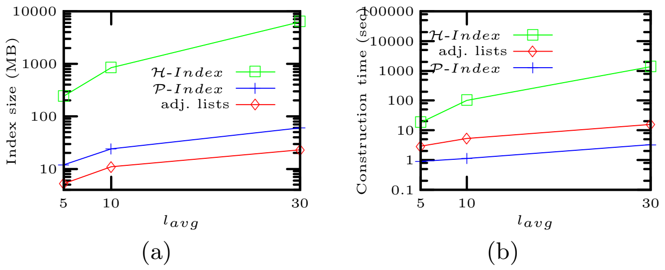


**Fig. 8.** (a) Index size, and (b) construction time varying $l_{avg}$, for $|\mathbf{P}| = 100000$, $|V| = 100000$, *zipf* $= 0.6$

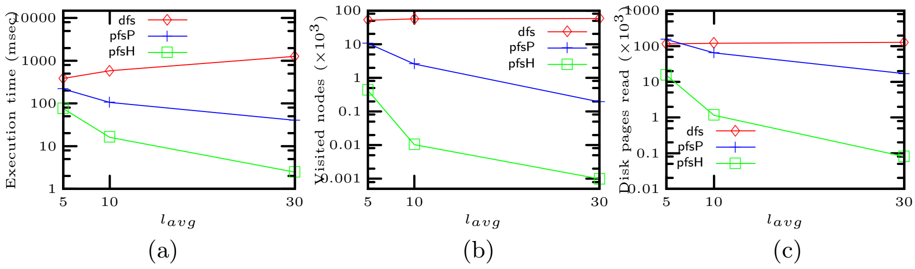


**Fig. 9.** (a) Average execution time, (b) average number of visited nodes, (c) average number of disk pages read varying $l_{avg}$, for $|\mathbf{P}| = 100000$, $|V| = 100000$, *zipf* $= 0.6$

to longer *edges* lists in $\mathcal{H}$-*Index* too, because there exist more common nodes between paths. Therefore, the space needed to store $\mathcal{P}$-*Index* and $\mathcal{H}$-*Index* also increases.

Figure 8(b) shows the effect on the construction time of the indices. The creation of all indices takes more time as $l_{avg}$ increases. Similarly to the case of varying $|\mathbf{P}|$, the construction time of the adjacency lists is higher than the time of $\mathcal{P}$-*Index*, since we first need to construct $G_{\mathbf{P}}$ graph by removing the repeated transitions between the nodes. The construction time of $\mathcal{H}$-*Index* is always at least one order of magnitude higher than the time of the other indices.

Figure 9(a) presents the effects of varying $l_{avg}$ on the query execution time. The experimental results show that the average execution time of pfsP and pfsH is lower than that of dfs in all cases. Moreover, as $l_{avg}$ increases, the execution time of dfs increases, whereas the execution time of pfsP and pfsH decreases. dfs becomes slower because the density of $G_{\mathbf{P}}$ increases. On the other hand, the join procedures in pfsP and pfsH will quickly identify a common path, since *paths* and *edges* lists become longer. Thus, both pfsP and pfsH need to visit fewer nodes to answer a query as Figure 9(b) shows. Figure 9(c) confirms the above findings with respect to the number of I/Os.

**Varying the number of nodes in the path collection.** Figure 10(a) illustrates the effects on the index size. As $|V|$ increases the adjacency lists and $\mathcal{P}$-*Index* require more disk space. In the case of the adjacency lists, this is because $G_{\mathbf{P}}$ becomes larger and more lists need to be stored. Similarly, the size of

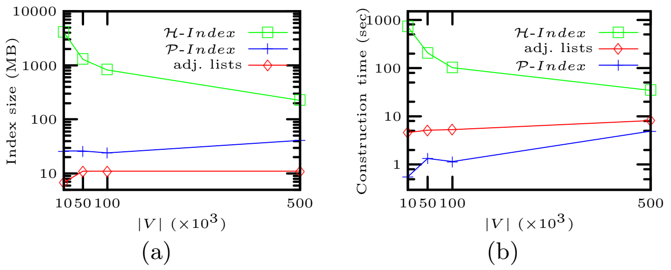


**Fig. 10.** (a) Index size, and (b) construction time varying $|V|$, for $|\mathbf{P}| = 100000$, $l_{avg} = 10$, $zipf = 0.6$

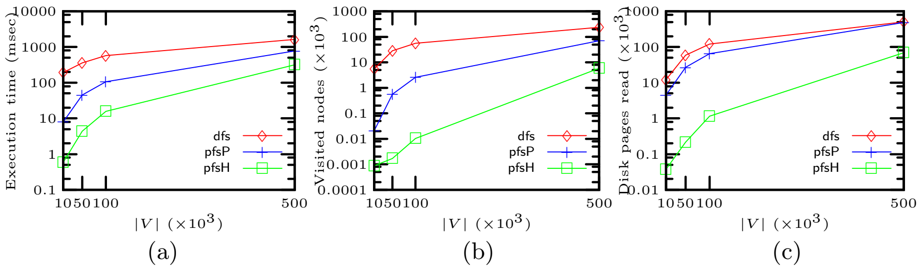


**Fig. 11.** (a) Average execution time, (b) average number of visited nodes, (c) average number of disk pages read varying $|V|$, for $|\mathbf{P}| = 100000$, $l_{avg} = 10$, $zipf = 0.6$

$\mathcal{P}$-*Index* also grows as $|V|$ increases, since it contains more *paths* lists. On the other hand, $\mathcal{H}$-*Index* requires less disk space as $|V|$ increases, because the paths have fewer common nodes and thus, the *edges* lists become shorter. Note that the total number of *edges* lists does not change as $|\mathbf{P}|$ is fixed to 100000.

Figure 10(b) shows the impact of varying $|V|$ on the construction time of the indices. As $|V|$ increases the construction of the adjacency lists of $G_\mathbf{P}$ and $\mathcal{P}$-*Index* takes more time. Similarly to the previous experiments, the construction time for the adjacency lists is higher since we need to construct $G_\mathbf{P}$ graph first. On the hand, the construction of $\mathcal{H}$-*Index* takes less time since the *edges* lists become shorter.

Figure 11(a) illustrates the effect of varying $|V|$ on the query execution time. All three algorithms are affected by the increase of $|V|$. Algorithms pfsP and pfsH are, in all cases, faster than dfs but the difference in the execution time decreases as $|V|$ increases. The performance of dfs is expected because $G_\mathbf{P}$ becomes larger as $|V|$ increases. Considering pfsP and pfsH, since (a) the collections include more nodes and (b) the number of paths is fixed, each node is contained in fewer paths and in addition, the paths have less nodes in common. In other words, since the *paths* lists of $\mathcal{P}$-*Index* and *edges* lists of $\mathcal{H}$-*Index* become shorter, they will likely have fewer common paths. Thus, both pfsP and pfsH need to visit more nodes to answer the queries as Figure 11(b) shows. Figure 11(c) confirms the above observations with respect to the number of I/Os.

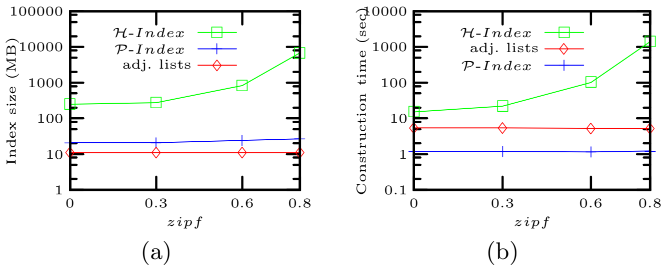


**Fig. 12.** (a) Index size, and (b) construction time varying *zipf*, for $|\mathbf{P}| = 100000$, $|V| = 100000$, $l_{avg} = 10$

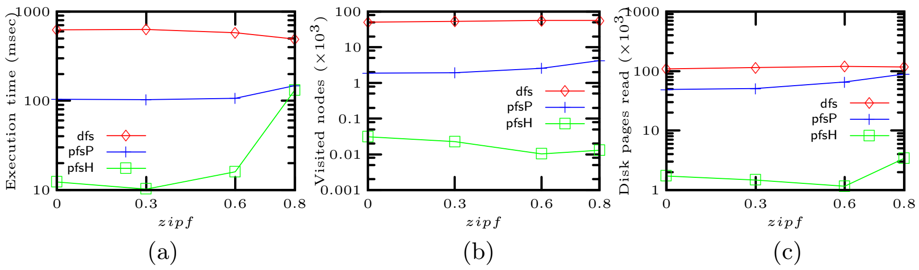


**Fig. 13.** (a) Average execution time, (b) average number of visited nodes, (c) average number of disk pages read varying *zipf*, for $|\mathbf{P}| = 100000$, $|V| = 100000$, $l_{avg} = 10$

**Varying node frequency in the path collection.** Figure 12(a) illustrates the effects on the index size. As expected, the increase of *zipf* does not affect the size of the adjacency lists. The total number of direct transitions between the nodes of the collection, i.e., the edges in $G_{\mathbf{P}}$ graph, does not change as *zipf* increases. The increase of *zipf* does not change the total number of entries of the *paths* lists in $\mathcal{P}$-*Index*, and therefore the size of $\mathcal{P}$-*Index* remains the same. On the other hand, the size of $\mathcal{H}$-*Index* increases. As *zipf* value increases some nodes can act as links for more paths of the collection. Thus, the *edges* lists become longer and the size of $\mathcal{H}$-*Index* increases.

Figure 12(b) shows the impact of varying the number of nodes in the collection on the construction time of the indices. As expected the construction time for the adjacency lists and $\mathcal{P}$-*Index* is not affected by the increase of *zipf*. In contrast, as $|V|$ increases the construction time of $\mathcal{H}$-*Index* increases.

Figure 13(a) shows the effect of varying *zipf* on the query execution time. We notice that the execution time of pfsP and pfsH is always lower than the execution time of dfs. Algorithm pfsH is faster than pfsP for approximately one order of magnitude for $zipf < 0.8$. As expected the execution time of dfs remains approximately stable since $G_{\mathbf{P}}$ does not change as *zipf* increases, whereas the execution time of pfsP and pfsH increases. The increase in the case of pfsP is less intense. Figure 13(b) shows that pfsP visits slightly more nodes as *zipf* increases. On the other hand, pfsH visits fewer nodes as *zipf* increases but retrieving the

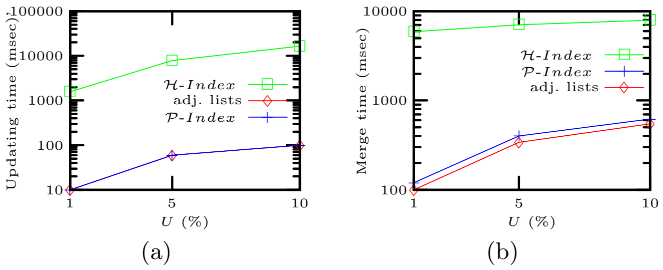


**Fig. 14.** (a) Updating time varying $U$, (b) index size varying $U$ for $|\mathbf{P}| = 100000$, $l_{avg} = 10$, $|V| = 100000$, $zipf = 0.6$

*edges* lists of the paths that contain very frequent nodes, costs a lot. Figure 13(c) confirms the above observations with respect to the number of I/Os.

**Updating path collections.** Finally, we study the methods for updating path collections. We measure: (a) the time required to update memory-based indices considering the new paths, and (b) the time needed to merge the disk-based indices with the memory-based ones.

Figure 14(a) illustrates the time required to update the memory-based indices. The updating time of $\mathcal{H}$-*Index* is higher than the time of the adjacency lists and $\mathcal{P}$-*Index* in all cases. This is due to the fact that we need to access the *edges* lists of the disk-based $\mathcal{H}$-*Index* to update the memory-based one. On the other hand, in all cases $\mathcal{P}$-*Index* and the adjacency lists are updated in equal time. Finally, Figure 14(b) shows that the time needed to merge the disk-based $\mathcal{H}$-*Index* is higher than the time required for the adjacency lists and $\mathcal{P}$-*Index*.

## 8 Conclusions

We consider reachability queries on path collections. We proposed the *path-first search* paradigm, which treats paths as first-class citizens, and further discussed appropriate indices that aid the search algorithms. Methods for updating a path collection and its indices were discussed. An extensive experimental evaluation verified the advantages of our approach. Our ongoing work focuses on compression techniques for $\mathcal{H}$-*Index*. In the future, we plan to extend our indexing methods to other types of queries, such as shortest path, nearest neighbor queries.

## References

1. Critchlow, T., Lacroix, Z.: Bioinformatics: Managing Scientific Data. Morgan Kaufmann, San Francisco (2003)
2. Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and distance queries via 2-hop labels. In: SODA, pp. 937–946 (2002)
3. Agrawal, R., Borgida, A., Jagadish, H.V.: Efficient management of transitive relationships in large data and knowledge bases. In: SIGMOD Conference, pp. 253–262 (1989)

4. Jin, R., Xiang, Y., Ruan, N., Wang, H.: Efficiently answering reachability queries on very large directed graphs. In: SIGMOD Conference, pp. 595–608 (2008)
5. Wang, H., He, H., Yang, J., Yu, P.S., Yu, J.X.: Dual labeling: Answering graph reachability queries in constant time. In: ICDE, p. 75 (2006)
6. Agrawal, R., Jagadish, H.V.: Materialization and incremental update of path information. In: ICDE, pp. 374–383 (1989)
7. Agrawal, R., Jagadish, H.V.: Direct algorithms for computing the transitive closure of database relations. In: VLDB, pp. 255–266 (1987)
8. Schenkel, R., Theobald, A., Weikum, G.: Efficient creation and incremental maintenance of the hopi index for complex xml document collections. In: ICDE, pp. 360–371 (2005)
9. Cheng, J., Yu, J.X., Lin, X., Wang, H., Yu, P.S.: Fast computing reachability labelings for large graphs with high compression rate. In: EDBT, pp. 193–204 (2008)
10. Trißl, S., Leser, U.: Fast and practical indexing and querying of very large graphs. In: SIGMOD Conference, pp. 845–856 (2007)
11. Bouros, P., Skiadopoulos, S., Dalamagas, T., Sacharidis, D., Sellis, T.K.: Evaluating reachability queries over path collections. Technical report, KDBS Lab, NTU Athens (2008), http://www.dblab.ece.ntua.gr/~pbour/reachOnPaths.pdf
12. Terrovitis, M.: Modelling and Operation Issues for Pattern Base Management Systems. PhD thesis, Knowledge and Database Systems Laboratory, School of Electrical and Computer Engineering, NTUA (May 2007)

# Easing the Dimensionality Curse by Stretching Metric Spaces

Ives R.V. Pola, Agma J.M. Traina, and Caetano Traina Jr.

Computer Science Department - ICMC
University of Sao Paulo at Sao Carlos – Brazil
{ives,agma,caetano}@icmc.usp.br

**Abstract.** Queries over sets of complex elements are performed extracting features from each element, which are used in place of the real ones during the processing. Extracting a large number of significant features increases the representative power of the feature vector and improves the query precision. However, each feature is a dimension in the representation space, consequently handling more features worsen the dimensionality curse. The problem derives from the fact that the elements tends to distribute all over the space and a large dimensionality allows them to spread over much broader spaces. Therefore, in high-dimensional spaces, elements are frequently farther from each other, so the distance differences among pairs of elements tends to homogenize. When searching for nearest neighbors, the first one is usually not close, but as long as one is found, small increases in the query radius tend to include several others. This effect increases the overlap between nodes in access methods indexing the dataset. Both spatial and metric access methods are sensitive to the problem. This paper presents a general strategy applicable to metric access methods in general, improving the performance of similarity queries in high dimensional spaces. Our technique applies a function that "stretches" the distances. Thus, close objects become closer and far ones become even farther. Experiments using the metric access method Slim-tree show that similarity queries performed in the transformed spaces demands up to 70% less distance calculations, 52% less disk access and reduces up to 57% in total time when comparing with the original spaces.

## 1 Introduction

Querying data repositories require defining the comparison criterion used in the operators that select the answer set. Simple data types, such as numbers and small character strings handled in current Database Management Systems (DBMS), assume a single, implicit, comparison criterion for each data domain: the magnitude order of the numbers and the lexicographical order of small strings. Those are valuable criteria, as they allow employing the total ordering (TO) property over the corresponding data domains. TO is the fundamental concept that allows applying the relational comparison operators ($<, \leq, >$ or $\geq$) over simple data types, besides the universally available equality ($=$ or $\neq$)

operators. It allows the construction of efficient access methods, such as the well-known B-tree and its variants, widely used in DBMS. However, more complex data, such as images, do not meet the TO property. Therefore, other comparison operators are required to retrieve such data types, and querying by similarity is the most suited option.

Images are a complex type of data that is broadly present in computational systems. The technique of querying image datasets based on its pictorial content is called Content-Based Image Retrieval (CBIR) [1] [2]. It is based on submitting each image to a set of image processing algorithms (called feature extractors) to obtain its feature vector, which are used in place of the real image to execute the comparison. For example, in generic CBIR systems, images are preprocessed to retrieve their color and/or texture histograms, polygonal contours of the pictured objects, etc.

Querying by similarity in such datasets requires defining a comparison criteria, which is usually expressed as result of the distance function between two feature vectors. In this paper, we employ the term *metric* for the distance function. The metric must hold some properties and is expected to return smaller values for more similar elements.

Formally, a metric space is a pair $< \mathbb{S}, d >$, where $\mathbb{S}$ is the domain of data elements and $d : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}^+$ is a metric holding the following properties, for any $s_1, s_2, s_3 \in \mathbb{S}$. Identity: $d(s_1, s_2) = 0 \Rightarrow s_1 = s_2$; Symmetry: $d(s_1, s_2) = d(s_2, s_1)$; Non-negativity: $0 < d(s_1, s_2) < \infty$ if $s_1 \neq s_2$; and Triangular inequality: $d(s_1, s_2) \leq d(s_1, s_3) + d(s_3, s_2)$.

Any dataset $S \subset \mathbb{S}$ associated to a metric is said to be in a metric space. The elements of a dataset $S$ can be composed of numbers, vectors, matrices, graphs and even functions. For example, images can be represented in a metric space using their feature vectors associated to a metric such as the Euclidean ($L_2$) or the Manhattan distance ($L_1$). There are studies aimed at developing extractors that obtain highly representative features from images and designing specific metrics in order to improve its precision [3,4].

Retrieving complex data by similarity is computationally expensive. Therefore, metric access methods (MAM) were developed employing the properties of metric spaces to index sets of complex data. A metric structure divides a dataset into regions and chooses representative objects to represent each region. Besides the objects, each node stores the representatives and their distances to the stored objects. Representatives are stored also in parent nodes, hierarchically organizing the structure as a tree. To answer a query, the query center is compared with the representatives in the root node. The triangular inequality is then used to prune subtrees, but as this property does not enforce total ordering, overlapping can occur.

It is well-known that a MAM's performance relies on how well it partitions the space, but it also depends on the behavior of the metric used [4]. A MAM is associated with a metric that measures the similarity between pairs of elements as the distance between them. The metric is then used to build and query the tree, making it to be referenced also as "distance-based trees".

It is natural to think that as more significant features are extracted, the higher the ability to correctly identify the images is, and more precise the answer will be. However, as more features are added to the feature vector, the larger dimensionality leads to the "dimensionality curse". This condition worsens the results and degrades the structure of the MAM [5]. Previous work targeted dimensionality reduction processes, but they are not useful for indexing, as they lead to distortions in the answers that are not easily predictable, preventing the use of correction techniques [6,7].

This paper proposes a technique to improve the performance of MAM by properly stretching the metric space, weakening the effect of the dimensionality curse. The technique employs a monotonically increasing function as a "*stretching function*", to change the space metric, producing a new field of distances. The field of stretched distances does not follow the properties of a metric space, but we show how to employ its properties to transform a MAM into an access method that perform similarity queries over the dataset using the distance field more efficiently than a MAM can do using the original space.

This paper is organized as follows. Section 2 shows the required concepts and related studies. Section 3 shows its motivation and describes the proposed technique. Section 4 shows the properties met by the stretched space that can be used to prune subtrees. Section 5 shows experiments and the results achieved. Finally, section 6 summarizes the ideas presented in this paper and draws its conclusions.

## 2 Background

Several Spatial Access Methods (SAMs) have been proposed for multidimensional data. A comprehensive survey showing the evolution of SAM and their main concepts can be found in [8]. However, the majority of them suffer from the dimensionality curse, having the performance drastically degenerated as the dimensionality increases [9,7,10]. Moreover, SAM are not applicable to non-dimensional data. In fact, SAM were first developed to deal efficiently with low dimensionality spaces, becoming inefficient as dimensionality raises above ten or so dimensions [7]. Hence, SAM are not useful to index feature vectors extracted from images, as they can have hundreds or thousands of dimensions.

A common practice to improve operations over data of large dimensionality is to employ the dimensionality reduction techniques. Most of the existing techniques rely on numeric processing or on neural networks, the majority of them considering all attributes at once [11] [12][13]. If small clusters occurring in the data depend on attributes that are meaningless for the bulk of the data, dimensionality reduction techniques might drop them, distorting the result.

Providing spatial data with a metric to enable handling them as a metric space helps reducing the problems derived from the dimensionality curse, because MAMs tend to follow the dimensionality of the object represented by the data (the so-called intrinsic dimensionality), not the dimensionality of the space where the object is embedded (the embedded dimensionality). Note that the intrinsic dimensionality is usually much smaller than the embedded one [7]. Moreover, many complex data do not have a defined dimensionality (e.g. polygons defining regions on images, because the number of edges varies), although a distance function can be defined to compare them. Thus, handling datasets of low intrinsic-dimensionality (or even non-dimensional ones) using MAMs rather than SAMs is an interesting way to speed up similarity queries. Two surveys on MAMs can be found in [14] and [15].

The M-tree [16] was the first balanced, dynamic metric tree proposed, followed by the Slim-tree [17], which includes the Slim-down algorithm to reduce node overlapping. The OMNI concept proposed in [18] increases the pruning power on searches by using strategically positioned elements, the foci set. These methods precalculate distances between selected elements, enabling the triangular inequality property to be used to prune subtree accesses during query evaluation.

The iDistance [19] is a $B^+$-tree based indexing method created to perform k-NN queries in high dimensional spaces. After partitioning the data space, a reference point is chosen for each partition, and the data points are transformed into a dimensional value, by using the similarity to the reference point, and indexed using a $B^+$-tree structure. Experiments comparing iDistance with other MAM suggests it is well suited for clustering detection and high-dimensional data.

The study of Nadvorny and Heuser [20] describes that the low query performance when querying text fields is a result from the achieved arrangement of the objects in the datasets over the metric space. This arrangement is inappropriate for grouping, and can jeopardize query performance, once the metric-tree structures are based on grouping objects. They propose applying a "twisting function" over the metric space, in order to generate a new space, where objects are better arranged. However, this new space does not allow queries once it does not have the metric properties, i.e., the triangular inequality (bounding property) does not hold. The partial solution became coming back to original space to do the queries. Although this approach enhances the data arrangement, most of the chosen functions are not monotonically crescent when applying a space mapping of distances. This condition makes it difficult deriving bounding properties in order to prune subtrees, and could lead to not preserving the order of distances from objects depending on the distribution of the dataset.

All of the above MAMs depend on the metric associated to the dataset. However, when the intrinsic dimensionality of the dataset is large, the dimensionality curse also affects them. Here, we state that the problem is not in the way MAMs organize themselves, but on the distribution of distances produced by the metrics associated to the datasets. The solution we propose acts at spreading the distribution of distances, aiming at expanding the metric space, improving the distance distribution in a way that enable a MAM to perform better.

## 3 Motivation

When there are many degrees of freedom, high dimensional data tend to spread out in the space. As a consequence, the distances between near and far objects become more similar, reducing the differences in distances occurring in the dataset.

In fact, in spaces of large dimensionality, elements tend to be far from each other, and as long as the nearest neighbor is reached, small increases in the covering radius encompass several other elements. This effect degenerates the MAMs, as they cannot partition the space adequately, increasing the node overlap. Both spatial and metric access methods are sensitive to this problem.

For the sake of clarity, we reproduce here an experiment described by Katayama and Satoh [21], which illustrates the effect of increasing the dimensionality over the distance distribution among elements of a dataset. The experiment consists of creating several datasets with the same cardinality, each having a different dimensionality, and measuring the minimum and the maximum distance between any pair of elements. Each element has a value randomly generated with uniform distribution in the range $[0, 1)$ in each dimension. Thus, each dataset corresponds to a set of points in a unit hyper-cube with the dimensionality of the dataset. We generated datasets with 100,000 points using 2, 4, 8, 16, 32, 64 and 128 dimensions. Figure 1 shows the minimum, maximum (diameter) and average distances between any pair of points in each dataset. As it can be seen, the minimum distance among elements of the 128-dimensional dataset is larger than half the measured dataset diameter.

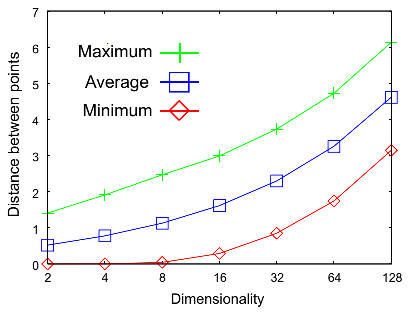


**Fig. 1.** Minimum, maximum and average distance between points in unit hyper-cubes of varying dimensionality

The effect is equivalent to place every element near to the border of the dataset, so no element has a really near neighbor. Executing similarity queries in those spaces is very expensive, and no index structure can significantly improve over sequential scan. Therefore, instead of trying to improve the access methods, we propose to change the space: our approach is to counteract the described effect developing a technique that stretches the metric space by the use of a "stretching function", in a way that the smaller distances are further compressed, and the larger ones are further expanded.

A stretching function is a monotonically increasing function $f : d \subseteq \mathbb{R}^+ \rightarrow \mathbb{R}^+$, where $d$ is a distance.

In order to illustrate the result of applying a stretching function, consider Figure 3. It shows a stretching function $f_s$ used to compare elements $(s_c, s_1, s_2, s_3)$, which are ordered by their distances to a center element $s_c$ through the metric $d$, which we call the original distances. The stretching function $f_s$ compresses/expands the original distance into a stretched distance field. To highlight the effect, the stretched distances produced by $f_s \circ d$ are shown in Figure 3 normalized to the largest distance among the elements. As it can be noticed, the elements that were close to $s_c$ become even closer and the elements that were far from $s_c$ become even farther. Therefore, applying the stretching function counteracts the effect of the dimensionality curse in high-dimensional spaces.

Employing a stretching function is equivalent to performing a space mapping: the stretching function maps the original field of distances centered at a given element into another field, which lead to a "distance-stretched space". Distinctly from dimensionality reduction techniques, a distance-stretched space preserves all the attributes in the feature vector, and all of them contribute equally to the data distribution.

It must be noted that a distance-stretched space is not metric. In fact, it is able to preserve the distance ordering of the original space considering only one center. However, the distance differences among pairs of objects that allow the objects to exchange positions are also governed by the stretching function. Therefore, it is possible to calculate the differences as an error when calculating

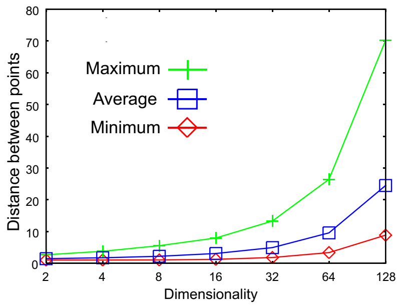


**Fig. 2.** Minimum, maximum and average distances stretched by function $f_s = e^d$

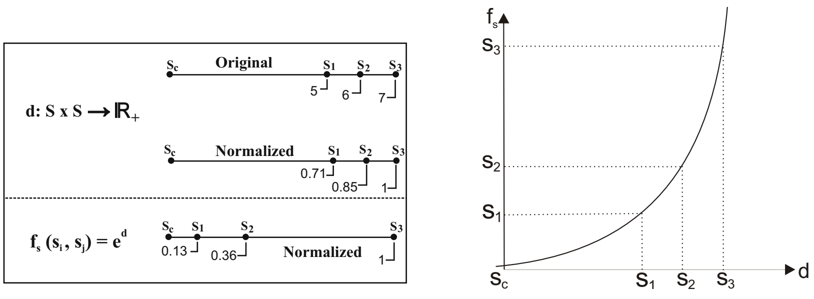


**Fig. 3.** The effect of stretching the space

the distances. A space mapping that guarantees a maximum error limit ($\alpha$) when mapping the distances from an original space into a target one is a well-known algebraic concept of the metric space theory, called a Linear $\alpha$-Lipschitz Continuous Mapping. In this paper, as we restrict the stretching function to be a strictly increasing continuous function, we analyze the function $f_s(d) = b^d$, where $b > 1$ is the stretching-base. In this way, the Lipschitz mapping defined for real valued distances is bounded by a constant defined as

$$\alpha \geq \frac{|b^{d_1} - b^{d_2}|}{|d_1 - d_2|}$$

for all $d_1, d_2 \in \mathbb{R}^+$ and $b > 1$.

To have an idea of how the distance distribution changes using a stretching function, Figure 2 shows the same experiment from Figure 1 now using the stretching function $f_s = e^d$ (that is, using the Euler's number $e$ as the stretching-base). As it can be noticed, the difference from the maximum to the minimum distances in the 128-dimensional dataset changes from half to less than 1/7, that is, similar to the distance distribution of the original 16-dimensional dataset.

The properties from the mapped space can be derived from those of the original ones. The next section presents the properties relevant to create index structures, detailing the derivation process, and presents the general formulation for any distance-stretched space modified by an exponential function.

## 4 Properties of Distance-Stretched Spaces

In this section we analyze the properties derived from a Linear $\alpha$-Lipschitz Continuous Mapping resulting from applying an exponential function to a metric $d$ (satisfying the metric axioms of identity, symmetry, non-negativity and triangular inequality described before), aimed at discovering properties useful for MAM to prune subtrees. Let us consider a stretching function defined as $f_s(s_i, s_j) = b^{d(s_i, s_j)}$, where $b \in \mathbb{R}^+, b > 1$ and $d$ is a metric. The resulting mapping produces a space that satisfies the properties as follows.

**Identity property** - In metric spaces, the identity property states that

$$d(s_i, s_j) = 0 \Rightarrow s_i = s_j$$

In the distance-stretched space, the identity property is transformed to the following expression

$$f_s(s_i, s_j) = 1 \Rightarrow s_i = s_j$$

because $b^0 = 1$

**Symmetry and non-negativity properties** - The original symmetry property holds in distance-stretched spaces too, once

$$f_s(s_i, s_j) = b^{d(s_i, s_j)} = b^{d(s_j, s_i)} = f_s(s_j, s_i)$$

as does the non-negativity property:

$$0 < f_s(s_i, s_j) = b^{d(s_i, s_j)} < \infty$$

**Bounding property** - The triangle inequality does not hold in distance-stretched spaces. To derive a useful rule, able to allow search algorithms to prune sub-trees, we use the triangular inequality property of the original spaces to set the lower and the upper bounds that the stretched distances must satisfy between pairs of any three elements. From the triangular inequality property, we can write

$$\underbrace{|d(s_1,s_3)-d(s_3,s_2)|}_{lower\ bound} \le d(s_1,s_2) \le \underbrace{d(s_1,s_3)+d(s_3,s_2)}_{upper\ bound}$$

for any $s_1, s_2, s_3 \in D$. Applying the stretching function, we can derive the bounding property generated by the stretching function $f_s(s_1,s_2) = b^{d(s_1,s_2)}$, which results in

$$b^{|d(s_1,s_3)-d(s_3,s_2)|} \le b^{d(s_1,s_2)} \le b^{d(s_1,s_3)+d(s_3,s_2)} \quad .$$

Defining $f_s^{-1}(s_i,s_j) = \log_b(f_s(s_1,s_2)) = d(s_i,s_j)$, we have

$$\underbrace{b^{|f_s^{-1}(s_1,s_3)-f_s^{-1}(s_3,s_2)|}}_{lower\ bound} \le f_s(s_1,s_2) \le \underbrace{b^{(f_s^{-1}(s_1,s_3)+f_s^{-1}(s_3,s_2))}}_{upper\ bound} .$$

This property gives us the lower and upper bound distances that can occur between any three elements in the distance-stretched space. This property can be employed to prune subtrees during retrieval operations in an index tree.

Although we can choose any type of exponential stretching function, a particular formulation can be written when applying to a specific stretching function. For example, without loss of generality, let us consider a stretching function of the form $f_s(s_i,s_j) = e^{d(s_i,s_j)}$. Then, the following properties hold in the distance field generated.

1. Identity: $f_s(s_i,s_j) = 1 \Rightarrow s_i = s_j$;
2. Symmetry: $f_s(s_i,s_j) = f_s(s_j,s_i)$;
3. Non-negativity: $0 < f_s(s_i,s_j) < \infty$;
4. Bounding: $e^{|ln(f_s(s_1,s_3))-ln(f_s(s_3,s_2))|} \le f_s(s_1,s_2) \le e^{(ln(f_s(s_1,s_3))+ln(f_s(s_3,s_2)))}$

## 4.1 Similarity Queries on Distance-Stretched Spaces

The similarity queries on distance-stretched spaces can take advantage of the bounding property in order to prune subtrees during the query execution. To employ the proposed technique over a MAM, it must be adapted so the stretched-distance space properties are employed in place of the metric ones. Although the changes are straightforward to implement, for the sake of completeness we present in this section the range query algorithm for the Slim-tree adapted to handle the stretched-distance space properties, which we call the Slim-tree XS.

We assume that the distances $d(s_p,s_i)$ from each element $s_i$ stored in a tree node to the node representative $s_p$ are stored in the node (following the structure of the Slim-tree, the M-tree, etc.), in order to reduce wasting time recalculating distances. So, in the formulas following, we mark the distances already stored as $\breve{d}$. To calculate the inverse mapping function is a costly operation too, so we propose to modify the index structure to store also the stretched value of the pre-calculated distances stored in the nodes, marking it as $\breve{f}_s$. Although this modification reduces the node capacity, it is shown in Section 5 that it is compensated by the higher pruning ability achieved by using the stretching functions.

For illustration purposes, Figure 4(b) exemplifies a query in a 2-dimensional Euclidean space. We use the notation $(s_i, r'_i)$ to represent a ball in a stretched-distance space centered at element $s_i$ and covering radius $r'_i$, where $r'_i = b^{r_i}$. In order to determine if the query ball $(s_q, r'_q)$ intersects the elements stored in a tree node covering the ball $(s_i, r'_i)$, we must evaluate if $f_s(s_i, s_q) \leq r'_i + r'_q$. If this condition is true, the balls intersect and the region of the ball $(s_i, r'_i)$ must be visited, like Figure 4(a) indicates. However, we also want to avoid the calculation of $f_s(s_i, s_q)$ using the lower bounding property and the stored distances. Evaluating if $b^{|\breve{f}_s^{-1}(s_i,s_p) - f_s^{-1}(s_p,s_q)|} > r'_i + r'_q$, we know that the query ball does not intersect the ball $(s_i, r'_i)$. It the test fails, the distance $f_s(s_i, s_q)$ must be calculated in order to determine if an intersection occurs.

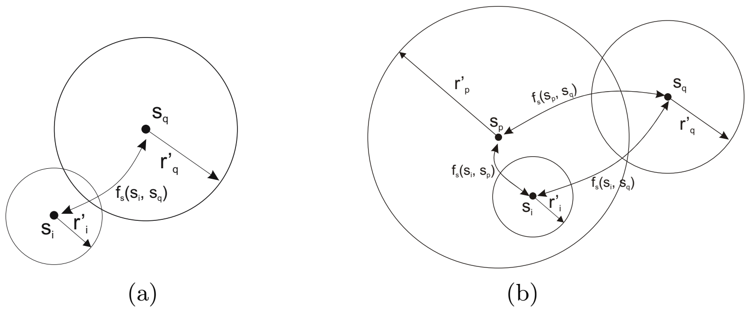


**Fig. 4.** (a) Intersection of balls in stretched space. (b) Example of a query $Rq(s_q, r'_q)$.

The range query algorithm for distance-stretched spaces is presented in Algorithm 1. The range query $Rq(s_q,\ r'_q)$ must select every element $s_j$ such that $f_s(s_j, s_q) < r'_q$. The algorithm starts from the root node and traverses every subtree that cannot be excluded by the bounding property, as indicated in Section 4.

A final note about the implementation details of the space-stretching technique regards the space normalization, as shown in Figure 3. Theoretically, normalization is not required, but it is worth remembering that an exponential function may generate very large values if the original distances are already large. Therefore, it is a good idea to sample a few elements in the original data space and use this reduced dataset to normalize the distances before applying the stretching function, to avoid numeric overflow. Notice that a rough approximation to a unit hypercube is enough as, unless preventing overflow, it has no effect in the final results.

**Algorithm 1.** Range Query ($s_q$, $r'_q$, *root*)

Input: query center $s_q$, query radius $r'_q$, root of subtree *node*.

```
Let $s_p$ be the parent element of *node*
**if** *node*.isLeaf() == false **then**
    **for all** $s_j$ in *node* **do**
        **if** $b^{|\check{f}_s^{-1}(s_i,s_p) - f_s^{-1}(s_p,s_q)|} \leq r'_i + r'_q$ **then**
            Evaluate $f_s(s_j, s_q)$;
            **if** $f_s(s_j, s_q) \leq r'_i + r'_q$ **then**
                Range Query($s_q$, $r'_q$, *node.subtree*($s_j$));
            **end if**
        **end if**
    **end for**
**else**
    **for all** $s_j$ in *node* **do**
        **if** $b^{|\check{f}_s^{-1}(s_i,s_p) - f_s^{-1}(s_p,s_q)|} \leq r'_q$ **then**
            Evaluate $f_s(s_j, s_q)$;
            **if** $f_s(s_j, s_q) \leq r'_q$ **then**
                Add *oid*($s_j$) to the result;
            **end if**
        **end if**
    **end for**
**end if**
```

## 5 Experiments and Results

This section presents the experiments performed to analyze the effect of stretching a metric space in metric access methods built over both synthetic and real world datasets. We used the Slim-tree MAM to measure the number of disk accesses, number of distances calculated and total time spent when performing similarity queries on both metric spaces and distance-stretched spaces.

The Slim-tree used is the one available at the Arboretum framework [1], which is written in C++. The tests were performed on a machine with a Pentium D 3.4GHz processor and 2Gb of memory RAM. The Slim-trees were built using the min-occupation and MST policies, which are considered as the best configuration by the authors.

For each dataset, two Slim-trees where built: one corresponding to the traditional tree storing the original metric space (referred here as the **Slim-tree**), and another corresponding to the modified tree storing the distance-stretched space (referred here as the **Slim-tree XS**). Notice that the only changes in the Slim-tree code are storing the stretched-distance version of every distance stored in a node and the corresponding use of these values in the query algorithms. In the experiments, the page size of the Slim-trees are set individually for each dataset, in order to have nodes with a maximum occupancy of 50 elements.

[1] http://gbdi.icmc.usp.br/arboretum

Each corresponding Slim-tree XS for the same dataset has the same page size of the Slim-tree, thus potentially leading to a maximum occupancy of slightly less than 50 elements in the Slim-tree XS.

All experiments were performed using a set of 500 queries with different query centers. For each experiment, it was measured the average number of disk accesses, the average number of distance calculations and the total time required to perform the 500 queries. The query elements are randomly chosen from those indexed in the structure.

Three kinds of datasets were used in the experiments, being one synthetic and two from the real world, as follows.

**Synthetic:** A set of seven synthetic datasets, each having 10,000 points, generated as a unit hypercube under a uniform distribution, with 2, 4, 8, 16, 32, 64 and 128 attributes (dimensions). We used the Euclidean metric for those datasets. Each dataset leads to the worse effects of the dimensionality curse in a dataset with the respective dimensionality, because no correlation exists among attributes.

**EnglishWords:** A random sample of 24,893 English words, using the $L_{edit}$ metric. This is also a hard case for similarity search, as the metric results in discrete values from a small cardinality codomain, leading to a large number of ties.

**PCA:** A set of 17,162 images projected on an orthonormal space of 43 eigenvectors (PCA) defined from a training set formed by linear sampling. We used the Euclidean metric for this dataset. This dataset exhibits an average behavior of real world datasets based on arrays of features, as its attributes are likely to present varying degrees of correlation.

## 5.1 Evaluating the Effect of the Dimensionality

The first experiment compares different stretching functions with the original space of distances when indexing the Synthetic datasets. We selected them to explore the high intrinsic dimension induced by the uniform distribution because, as there is no correlation among attributes, the dimensionality curse will be the strongest, hurting the performance of indexing structures the most.

We tested the functions

**Power2:** $f_s(s_i, s_j) = 2^{d(s_i,s_j)}$
**PowerE:** $f_s(s_i, s_j) = e^{d(s_i,s_j)}$
**Power10:** $f_s(s_i, s_j) = 10^{d(s_i,s_j)}$

composed with the Euclidean metric ($L_2$).

In order to analyze when the dimensionality curse takes place in similarity queries, we performed $k$-nearest neighbor queries ($k$-NNq) and range queries (Rq) indexing the Synthetic datasets with the Slim-tree and the Slim-tree XS, and evaluated their behavior as the dimensionality increases. The selected number of nearest neighbors for $k$-NNq is $k = 6$ and the covering radius for Rq was set to $radius = 0.01$ (1% of the maximum dataset span in a dimension).

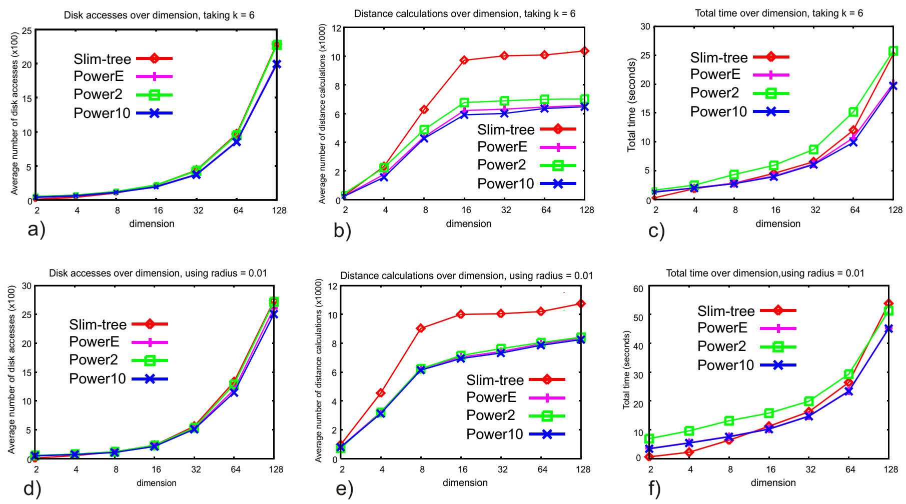


**Fig. 5.** Evaluating different stretching functions on similarity queries over increasing dimensionality of the Synthetic datasets. Plots a), b and c) correspond to $k$-NN queries and plots d), e) and f) to range queries. Plots a) and d) show the average number of disk accesses, plots b) and e) show the average number of distance calculations, and plots c) and f) show the total evaluation time required to perform 500 queries.

The results are presented in Figure 5. Notice that both $k$-nearest (plots a),b),c) at top line) and range queries plots (d),e),f) at lower line) follow the same pattern. Regarding the number of distance calculations, Figures 5b) and 5e) show that the distance-stretched space eases the dimensionality curse. In fact, after 16 dimensions, the stretched space provides a steady reduction of approximately 40% of the number of distance calculations for both k-NNq and range queries, regardless of the stretching-base used. We also can see that, although increasing the stretching-base reduces the required number of distance calculations, a small base is enough to provide good reduction, and that after a base equal to the Euler's number $e$, further reductions becomes small.

Regarding the number of disk accesses, Figures 5a) and 5d) show that the number of disk accesses required in the distance-stretched space is close but always smaller than in the original space. This occurs likely because of the extra distance stored in the node for each element. Thus, the Slim-tree XS storing the distance-stretched space has more nodes than the Slim-tree in the original space. However, even with more nodes, the figures show that the Slim-tree XS in the distance-stretched space performs less disk accesses.

Figures 5c) and 5f) show the time spent to execute the queries. They show that a stretching-base $b = e$ or $b = 10$ leads to almost the same timing, whereas $b = 2$ leads to the longest time. This is probably due to the large time required to process the base 2 logarithm. It can also be seen that querying the original space for datasets up to 4 dimensions is faster than querying the distance-stretched space (probably due to not using the exponential and logarithm functions), but at eight or more dimensions, querying the distance-stretched space is faster.

Figure 5 shows that different stretching functions produce different distance spaces with slightly variations on performance. Although it is intuitive that functions $b^d$ with a higher base $b$ better distribute the distances between pairs of elements, there is a performance penalty based on the algorithm employed to calculate both the exponential and, more costly, its inverse function $\log_b d$. As we can see, the larger the value of the base, the better the improvement achieved, although using a stretching-base larger than $e$ leads to small gain. Moreover, one must care for the value of the base, because if the distances among pairs of objects are large, the stretching function overflows. Therefore, this experiment shows that using $e$ as the stretching-base is a good choice, and the next experiments were performed using only the stretching function $e^d$.

## 5.2 Evaluating the Performance of Range and k-NN Queries

The second experiment measures the performance of similarity queries over the *EnglishWords* and the *PCA* datasets. Table 1 shows the number of distance calculations and the time spent to construct the Slim-trees and the Slim-tree

**Table 1.** Construction statistics for the real datasets

| | PCA | | EnglishWords | |
|---|---|---|---|---|
| | Distances | Time(ms) | Distances | Time(ms) |
| Slim-tree | 1,209,833 | 3,640 | 1,882,419 | 9,437 |
| Slim-tree XS | 1,160,576 | 2,906 | 1,605,959 | 7,906 |

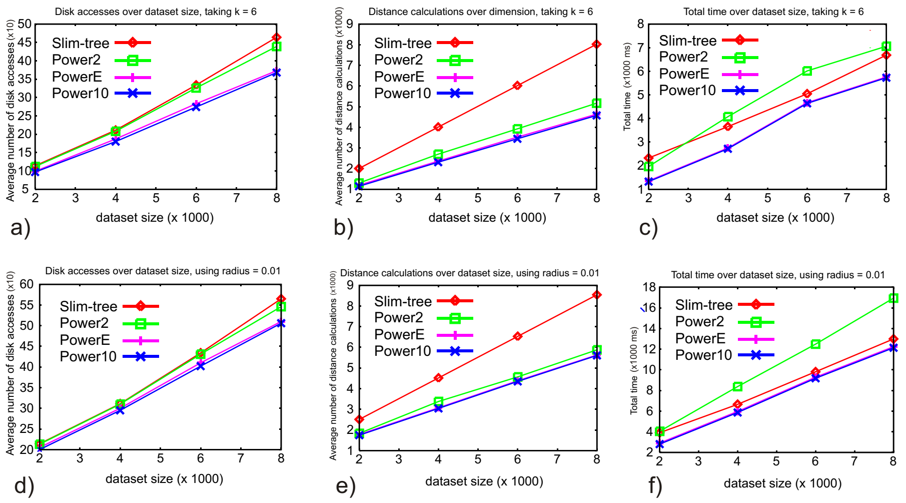


**Fig. 6.** Behavior of different stretching functions on similarity queries over increasing size of the Synthetic dataset. The first line corresponds to $k$-NN queries and the second to range queries. The first column shows the average number of disk accesses, the second one shows the average number of distance calculations, and the third one shows the total evaluation time required to perform 500 queries.

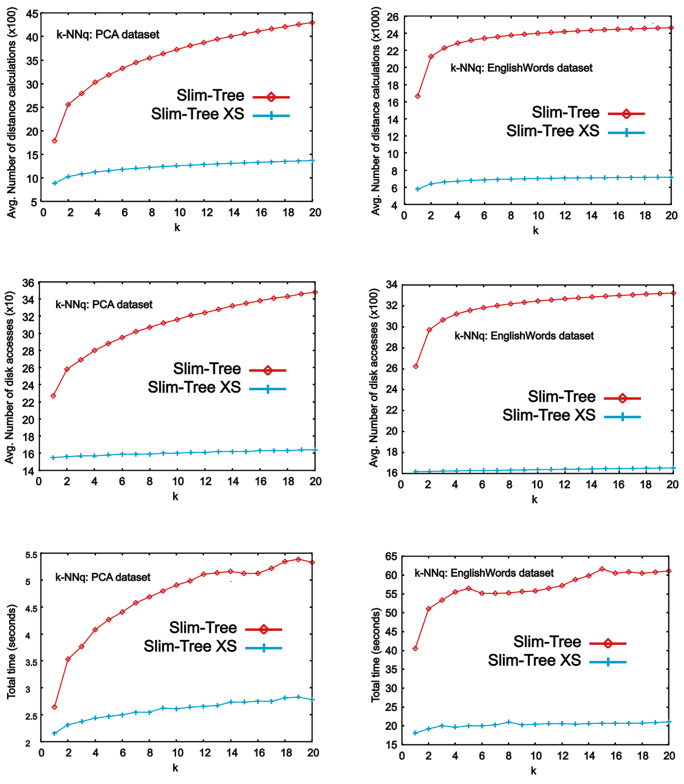


**Fig. 7.** K Nearest Neighbor queries measures over the datasets PCA and EnglishWords

XS over both datasets. As we can see, constructing the Slim-trees XS required less distance calculations and a shorter time for both datasets. Besides faster to construct, the Slim-tree XS trees also enable faster query processing, as the following experiments show.

In this experiment we also measured the average number of distances calculated, the average number of disk accesses and the total time spent to answer 500 queries. Figure 7 shows the results. The radius of the range queries, presented in logarithmic scale in plots, is the percentile of similar elements obtained by the query, i. e., a range query of radius 0.1 retrieves approximately 10% of the elements from the dataset.

Analyzing the graphs of queries executed over the PCA dataset (the first two columns of Figure 7), we can see that the Slim-tree XS achieved better performance than the classical Slim-tree. In fact, the Slim-tree performed three times more distance calculations in average than the Slim-tree XS (the first line of plots in Figure 7). Regarding the number of disk accesses, both range and $k$-NN queries presented the same behavior, and the regular Slim-tree performed more than twice disk accesses, in average, than the Slim-tree XS (second line of Figure 7). Considering the time spent, the Slim-tree XS also outperformed the Slim-tree, being, in average, twice as faster as the Slim-tree.

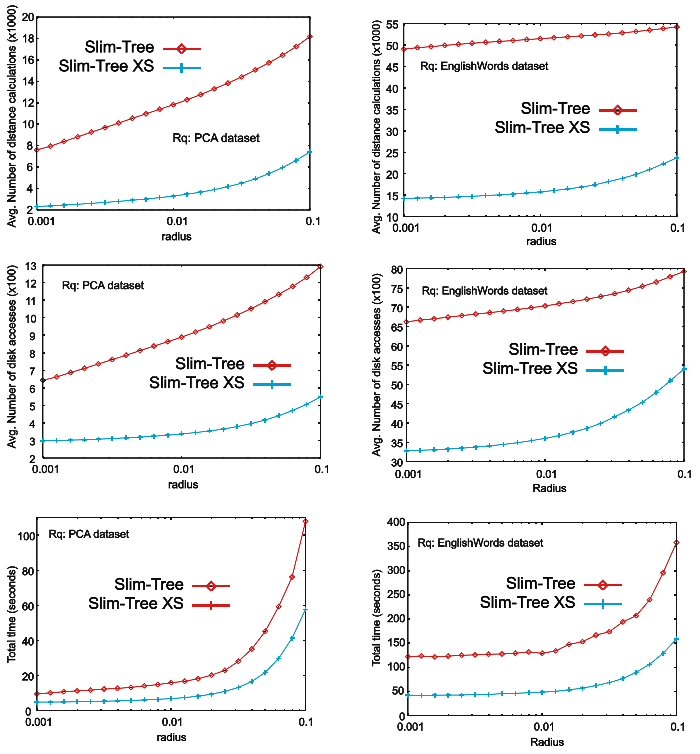


**Fig. 8.** Range queries measures over the datasets PCA and EnglishWords

Another interesting result is the performance of the Slim-tree XS indexing the EnglishWords dataset. This dataset is adimensional, and the $L_{Edit}$ metric returns integer values varying from 0 (for identity) up to 24 (the maximum number of letters in a English word in the dataset). Again, we can see that the Slim-tree XS exhibits better performance gains against the Slim-tree, as the Slim-tree required in average 3.5 more distances calculations and twice the number of disk accesses to perform both range and $k$-NN queries than the Slim-tree XS. The Slim-tree XS was also at least 2.5 times faster than the Slim-tree for both kinds of similarity queries.

## 5.3 Evaluating Scalability

The last experiment evaluates the scalability of the proposed technique. The experiment performs range and $k$-nearest neighbor queries over Synthetic dataset with 16 dimensions of increasing sizes indexed by both Slim-tree and Slim-tree XS. We performed queries using different stretching functions, measuring the average number of disk accesses, the average number of distance calculations and the total time required to execute 500 queries.

Figure 6 shows the results obtained when performing range and $k$-nearest neighbor queries with increasing dataset size. The graphs show that the indexing methods scales linearly with the size of the datasets, regardless of the stretching-base employed. It also confirms that using a stretching base $b > e$ is enough to obtain a steady performance gain.

## 6 Conclusion

Many applications deal with complex data, where a large number of features, or dimensions are necessary to represent the essence of the data contents. Thus, these data are inherently high-dimensional, what leads to the problem of the dimensionality curse when processing similarity queries. Consequently new techniques are required to avoid this curse, which really damages the managing of the information. Previous work targeted the dimensionality reduction processes. However, the majority of them relies on changing the attributes of the data space. Therefore, they are not adequate for indexing purposes supporting similarity search, once they can cause distortions in the answers that are not easily predictable.

In this paper we introduced a new mapping technique, which requires only minor changes in existing metric access methods, and greatly reduces the negative effects of the dimensionality curse. Its main idea is to stretch the metric space following an exponential function, in order to push the elements far from the query center even farther, whereas close elements are pulled even closer. As the stretching function modify the space of distances between pairs of elements, new properties were derived from the axioms of metric spaces. They are used in place of the metric ones in order to prune subtrees in indexing structures when performing queries.

Experiments were made to evaluate query execution on the distance-stretched space using the Slim-tree access method. However, the proposed method can be straightforwardly applied to any MAM. The reformulation of a Slim-tree to work on distance-stretched spaces supporting the proposed properties was called Slim-tree XS.

Experiments made on synthetic datasets with up to 128 dimensions showed that the Slim-tree XS achieved better performance than Slim-tree as the dataset dimensionality increases. Results on $k$-NN queries pointed that the Slim-tree XS achieved up to 43% less distance calculations and up demands to 7% less disk accesses, resulting in a gain in time of 20%. Results on range queries pointed that the Slim-tree XS achieved up to 25% less distance calculations and up to 4% less disk accesses, resulting in a gain in time of 18%.

Experiments made on real datasets showed that the mapping technique improves the performance of similarity queries on high dimensional, real world datasets, such as image features extracted with PCA methods and even on adimensional datasets, such as a set of words from the English language. Considering both datasets, results on $k$-NN queries pointed that the Slim-tree XS achieved up to 70% less distance calculations, up to 52% less disk accesses and a gain in time of 57% as compared to the plain Slim-tree. Results on range queries showed that the Slim-tree XS achieved up to 58% less distance calculations and up to 45% less disk accesses, resulting in a gain in time of 51%. Finally, scalability analysis made on *Synthetic* datasets showed a that the technique scales linearly as the dataset size increases.

Thus, we claim that the proposed distance-stretching technique is a novel and very effective approach to extend the database technology to meet the increasing need to support high dimensional objects in DBMS.

## Acknowledgments

This research was partially supported by CNPq, CAPES and by FAPESP.

## References

1. Smeulders, A.W.M., Worring, M., Santini, S., Gupta, A., Jain, R.: Content-based image retrieval at the end of the early years. IEEE-PAMI 22(12), 1349–1380 (2000)
2. Güntzer, U., Balke, W.T., Kiessling, W.: Optimizing multi-feature queries for image databases. In: VLDB, Cairo - Egypt, pp. 419–428 (2000)
3. Felipe, J.C., Traina, A.J.M., Caetano Traina, J.: Global warp metric distance: Boosting content-based image retrieval through histograms. In: ISM 2005: Proceedings of the Seventh IEEE International Symposium on Multimedia, Washington, DC, USA, pp. 295–302. IEEE Computer Society, Los Alamitos (2005)
4. Bugatti, H.P., Traina, A.J.M., Traina, C.J.: Assessing the best integration between distance-function and image-feature to answer similarity queries. In: 23rd Annual ACM Symposium on Applied Computing (SAC 2008), Fortaleza, Ceará - Brazil, pp. 1225–1230. ACM Press, New York (2008)
5. Beyer, K., Godstein, J., Ramakrishnan, R., Shaft, U.: When is "nearest neighbor" meaningful? In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 217–235. Springer, Heidelberg (1998)
6. Aggarwal, C.C., Hinneburg, A., Keim, D.A.: On the surprising behavior of distance metrics in high dimensional spaces. In: Van den Bussche, J., Vianu, V. (eds.) ICDT 2001. LNCS, vol. 1973, pp. 420–434. Springer, Heidelberg (2001)
7. Korn, F., Pagel, B.U., Faloutsos, C.: On the 'dimensionality curse' and the 'self-similarity blessing'. IEEE Transactions on Knowledge and Data Engineering (TKDE) 13(1), 96–111 (2001)
8. Gaede, V., Günther, O.: Multidimensional access methods. ACM Computing Surveys 30(2), 170–231 (1998)
9. Berchtold, S., Böhm, C., Kriegel, H.P.: The pyramid-tree: Breaking the curse of dimensionality. In: ACM SIGMOD International Conference on Management of Data, Seattle, WA, pp. 142–153 (1998)
10. Yianilos, P.N.: Locally lifting the curse of dimensionality for nearest neighbor search. In: Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 361–370 (2000)
11. Papamarkos, N., Atsalakis, A.E., Strouthopoulos, C.P.: Adaptive color reduction. IEEE Transactions on Systems, Man and Cybernetics 32(1), 44–56 (2002)

12. Park, M., Jin, J.S., Wilson, L.S.: Fast content-based image retrieval using quasi-gabor filter and reduction of image feature dimension. In: SSIAI 2002, Santa Fe, New Mexico, pp. 178–182. IEEE Computer Society, Los Alamitos (2002)
13. Ye, J., Li, Q., Xiong, H., Park, H., Janardan, R., Kumar, V.: Idr/qr: An incremental dimension reduction algorithm via qr decomposition. TKDE 17(9), 1208–1222 (2005)
14. Chávez, E., Navarro, G., Baeza-Yates, R.A., Marroquín, J.L.: Searching in metric spaces. ACM Computing Surveys 33(3), 273–321 (2001)
15. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces. ACM-TODS 21(4), 517–580 (2003)
16. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: Jarke, M. (ed.) VLDB, Athens, Greece, pp. 426–435. Morgan Kaufmann, San Francisco (1997)
17. Traina Jr., C., Traina, A.J.M., Faloutsos, C., Seeger, B.: Fast indexing and visualization of metric datasets using slim-trees. IEEE Transactions on Knowledge and Data Engineering (TKDE) 14(2), 244–260 (2002)
18. Santos Filho, R.F., Traina, A.J.M., Traina Jr., C., Faloutsos, C.: Similarity search without tears: The omni family of all-purpose access methods. In: ICDE, Heidelberg, Germany, pp. 623–630. IEEE Computer Society Press, Los Alamitos (2001)
19. Jagadish, H.V., Ooi, B.C., Tan, K.L., Yu, C., Zhang, R.: idistance: An adaptive b+-tree based indexing method for nearest neighbor search. TODS 30(1), 364–397 (2005)
20. Nadvorny, C.F., Heuser, C.A.: Twisting the metric space to achieve better metric trees. In: SBBD, pp. 178–190 (2004)
21. Katayama, N., Satoh, S.: Distinctiveness-sensitive nearest neighbor search for efficient similarity retrieval of multimedia information. In: ICDE, Washington, DC, USA, pp. 493–502. IEEE Computer Society Press, Los Alamitos (2001)

# Probabilistic Similarity Search for Uncertain Time Series

Johannes Aßfalg, Hans-Peter Kriegel, Peer Kröger, and Matthias Renz

Ludwig-Maximilians-Universität München, Oettingenstr. 67, 80538 Munich, Germany
{assfalg,kriegel,kroegerp,renz}@dbs.ifi.lmu.de
http://www.dbs.ifi.lmu.de

**Abstract.** A probabilistic similarity query over uncertain data assigns to each uncertain database object *o* a probability indicating the likelihood that *o* meets the query predicate. In this paper, we formalize the notion of uncertain time series and introduce two novel and important types of probabilistic range queries over uncertain time series. Furthermore, we propose an original approximate representation of uncertain time series that can be used to efficiently support both new query types by upper and lower bounding the Euclidean distance.

## 1 Introduction

Similarity search in time series databases is an active area of research usually with a focus on certain data. No work has been done so far to support query processing on uncertain time series. Uncertainty is important in emerging applications dealing e.g. with moving objects or object identification as well as sensor network monitoring. In all these applications, the observed values at each time slot of a time series exhibit various degrees of uncertainty. Due to the uncertainty of the data objects, similarity queries are probabilistic rather than exact: we can only assign to each database object a probability that it meets the query predicate. As a consequence, there is a need to adapt storage models and indexing/search techniques to deal with uncertainty [1,2,3,4]. Furthermore several approaches for probabilistic query processing have been proposed recently including probabilistic range queries [5,6], probabilistic *k*NN and top-*k* queries [7,8,9,2,10] and probabilistic ranking [10,11,12,13,14]. Applications where the analysis of time series has to cope with uncertainty are e.g. traffic measurements in road networks, location tracking of moving objects or measuring environmental parameters as temperature.

When looking at the above sketched applications, we can extract two types of uncertain time series model uncertainty using a sampling approach rather than probability density functions (pdfs). In the first two applications, the sample values of different time slots are uncorrelated, i.e. there is no relationship between a given sample observation at time slot *i* and another sample observation at time slot $(i+1)$. On the other hand, in Application 2, each observed sample at time slot *i* is correlated to an observation at time slot $(i+1)$ and *vice versa*. Since both types require different and complex solutions in order to support probabilistic similarity queries, we only focus on uncorrelated uncertain time series throughout the rest of the paper. As indicated above, we assume that uncertainty is modelled using sample observations rather than pdfs.

To the best of our knowledge, this is the first paper, that formalizes the problem of probabilistic queries on uncertain time series, focusing on two types of probabilistic range queries (cf. Sec. 2). Furthermore, this paper proposes a novel compact approximation of uncertain time series and shows how upper and lower bounding distance estimations for Euclidean distance can be derived from these representations (cf. Sec. 3). Third, it illustrates how these distance approximations can be used to implement a multi-step query processor answering probabilistic similarity queries on uncertain time series efficiently (cf. Sec. 3).

## 2 Probabilistic Queries over Uncertain Time Series

Usually, time series are sequences of (certain) $d$-dimensional points. Uncertain time series are sequences of points having an uncertain position in the $d$-dimensional vector space. This uncertainty is represented by a set of sample observations at each time slot.

**Definition 1 (Uncertain Time Series).** *An* uncertain time series $\mathcal{X}$ *of length* $n$ *consists of a sequence* $\langle X_1, \ldots, X_n \rangle$ *of* $n$ *elements, where each element* $X_t$ *contains a set of* $s$ $d$*-dimensional points (sample observations), i.e.* $X_t = \{x_{t,1}, \ldots, x_{t,s}\}$ *with* $x_{t,i} \in \mathbb{R}^d$. *We call* $s$ *the* sample size *of* $\mathcal{X}$. *The distribution of the points in* $X_t$ *reflects the uncertainty of* $\mathcal{X}$ *at time slot* $t$.

We will use the term *regular time series* for traditional, non-uncertain (i.e. exact) time series consisting of only one $d$-dimensional point at each time slot[1].

In order to measure the similarity of uncertain time series we need a distance measure for such uncertain time series. For regular time series, e.g. any $L_p$-norm is commonly used to measure the distance between pairs of time series. Due to the uncertainty of the time series, also the distance between two time series is uncertain. Instead of computing one unique distance value such as the $L_p$-norm of the corresponding sequences, the distance between uncertain time series rather consists of multiple distance values reflecting the distribution of all possible distance values between the samples of the corresponding uncertain time series. This intuition is formalized in the following definition.

**Definition 2 (Uncertain $L_p$-Distance).** *For a one-dimensional uncertain time series* $\mathcal{X}$ *of length* $n$, *let* $s_{\mathcal{X}}$ *be the sample size of* $\mathcal{X}$ *and* $TS_{\mathcal{X}}$ *be the set of all possible regular time series that can be derived from the combination of different sample points of* $\mathcal{X}$ *by taking one sample from each time slot, i.e.*

$$TS_{\mathcal{X}} = \{\langle x_{1,1}, x_{2,1}, \ldots, x_{n,1} \rangle, \ldots, \langle x_{1,s_{\mathcal{X}}}, x_{2,s_{\mathcal{X}}}, \ldots, x_{n,s_{\mathcal{X}}} \rangle\}.$$

[1] For presentation issues, we assume 1-dimensional uncertain time series, the extension to the general $d$-dimensional case is straightforward.

*The $L_p$-distance between two uncertain time series $\mathcal{X}$ and $\mathcal{Y}$, denoted by $\widetilde{dist}_p$, is a collection containing the $L_p$ distances of all possible combinations from $TS_\mathcal{X}$ and $TS_\mathcal{Y}$, i.e. $\widetilde{dist}_p(\mathcal{X},\mathcal{Y}) = \{L_p(x,y)\,|\,x \in TS_\mathcal{X}, y \in TS_\mathcal{Y}\}$.*

Based on the distance function $\widetilde{dist}_p$ we define two query types for uncertain time series. Thereby, we define the probability $\Pr(\widetilde{dist}_p(\mathcal{X},\mathcal{Y}) \le \varepsilon)$ that the distance between two uncertain time series $\mathcal{X}$ and $\mathcal{Y}$ is below a given threshold $\varepsilon$ as

$$\Pr(\widetilde{dist}_p(\mathcal{X},\mathcal{Y}) \le \varepsilon) = \frac{|\{d \in \widetilde{dist}_p(\mathcal{X},\mathcal{Y}) | d \le \varepsilon\}|}{s_\mathcal{X}^n \cdot s_\mathcal{Y}^n}.$$

**Definition 3 (Probabilistic Bounded Range Query).** *Let $\mathcal{D}$ be a database of uncertain time series, $\varepsilon \in \mathbb{R}^+$, and $\tau \in [0,1]$. For an uncertain time series $\mathcal{Q}$, the* Probabilistic Bounded Range Query *(PBRQ) returns the following set*

$$RQ_{\varepsilon,\tau}(\mathcal{Q},\mathcal{D}) = \{\mathcal{X} \in \mathcal{D}\,|\, \Pr(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \le \varepsilon) \ge \tau\}.$$

**Definition 4 (Probabilistic Ranked Range Query).** *Let $\mathcal{D}$ be a database of uncertain time series and $\varepsilon \in \mathbb{R}^+$. For an uncertain query time series $\mathcal{Q}$, the* Probabilistic Ranked Range Query *(PRRQ) returns an ordered list:*

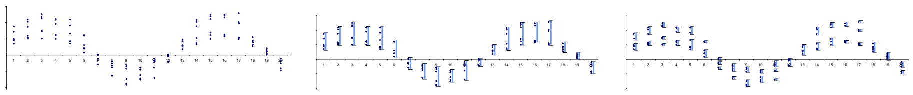
$$RQ_{\varepsilon,rank}(\mathcal{Q},\mathcal{D}) = (\mathcal{X}_1, \ldots, \mathcal{X}_m),$$

*where* $\Pr(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}_i) \le \Pr(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}_{i+1})$ *(*$1 \le i \le, m-1$*) and* $\Pr(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}_i)$ $\le \varepsilon)$ *for all $i = 1,\ldots,m$. For efficiency reasons, we assume a function* getNext *on the set $RQ_{\varepsilon,rank}(\mathcal{Q},\mathcal{D})$ that returns the next element of the ranking, i.e. the first call of* getNext *returns the first element in $RQ_{\varepsilon,rank}(\mathcal{Q},\mathcal{D})$, the second call returns the second element in $RQ_{\varepsilon,rank}(\mathcal{Q},\mathcal{D})$, and so on.*

Let us note that in the database context where we have long time series (high value of $n$) and high sample rates, the naive solution for both query types are CPU-bound because for all $\mathcal{X} \in \mathcal{D}$ we need to compute all distance observations in $\widetilde{dist}_p(\mathcal{Q},\mathcal{X})$ in order to determine $\Pr(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \le \varepsilon)$. This means that a naive solution requires to compute for each $\mathcal{X} \in \mathcal{D}$ exactly $|\widetilde{dist}_p(\mathcal{Q},\mathcal{X})| = s_\mathcal{Q}^n \cdot s_\mathcal{X}^n$ distances. For large values of $n$, $s_\mathcal{Q}$, and $s_\mathcal{X}$, this is obviously much more costly than sequentially scanning the disk to access all $\mathcal{X} \in \mathcal{D}$.

## 3 Multi-step Probabilistic Range Query Processing

Obviously, the CPU cost (and thus, the overall runtime) of our probabilistic similarity queries are dominated by the number of distance calculations necessary to determine the probability $\Pr(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \le \varepsilon)$ for a query object $\mathcal{Q}$ and all $\mathcal{X} \in \mathcal{D}$. This high number results from the combination of the observed distance values between $\mathcal{Q}$ and $\mathcal{X}$ at each time slot. A first idea for runtime reduction is that we only need to determine the number of distance observations



(a) Exact level. (b) First level. (c) Second level.

**Fig. 1.** Different levels of approximating uncertain time series

$d \in \widetilde{dist}_p(\mathcal{Q}, \mathcal{X})$ with $d \leq \varepsilon$ because $|\widetilde{dist}_p(\mathcal{Q}, \mathcal{X})| = s^n_{\mathcal{Q}} \cdot s^n_{\mathcal{X}}$. We can further improve the runtime by calculating lower and upper bounds for the probability that further reduce the number of distance computations. For that purpose, we have to calculate an upper and a lower bound for the number of distance observations $d \in \widetilde{dist}_p(\mathcal{Q}, \mathcal{X})$ with $d \leq \varepsilon$.

## 3.1 Approximative Representation

Intuitively, we construct the approximative representation of an uncertain time series $\mathcal{X}$ by aggregating the observations $x_{i,j} \in X_i$ at each time slot $i$ into groups and use these groups to calculate the distance between uncertain time series. Obviously, this reduces the sample rate and thus, the overall number of possible distance combinations. The groups are represented by minimum bounding intervals[2].

**Definition 5 (Approximative Representation).** *The* approximative representation $\mathcal{X}_a$ *of an uncertain time series* $\mathcal{X}$ *of length* $n$ *consists of a sequence* $\langle\{I_{1,1}, \ldots, I_{1,m_1}\}, \ldots, \{I_{n,1}, \ldots, I_{n,m_n}\}\rangle$ *of interval sets. Each interval* $I_{i,j} = [l_{i,j}, u_{i,j}]$ *minimally covers a given number* $|I_{i,j}|$ *of sample points of* $X_i$*, i.e.* $l_{i,j}$ *and* $u_{i,j}$ *are sample points of* $X_i$*, at time slot* $i$.

We use two levels of approximation. The first level describes all sample points at time slot $i$ by one minimal bounding interval (cf. Figure 1(b)), i.e. $m_i = 1$ for all time slots $i$ and $\mathcal{X}_a = \langle I_{1,1}, \ldots, I_{n,1}\rangle$. For the second level approximations, the sample observations at time slot $i$ are grouped into $k$ clusters by applying the algorithm $k$-means [15] on all $x_{i,j} \in X_i$ (cf. Figure 1(c)), i.e. $m_i = k$ for all time slots $i$ and $\mathcal{X}_a = \langle\{I_{1,1}, \ldots, I_{1,k}\}, \ldots, \{I_{n,1}, \ldots, I_{n,k}\}\rangle$.

## 3.2 Distance Approximations

Using approximative representations $\mathcal{X}_a$ and $\mathcal{Y}_a$ of two uncertain time series $\mathcal{X}$ and $\mathcal{Y}$ we are able to calculate lower and upper bounds for $\Pr(\widetilde{dist}_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon)$.

Analogously to Definition 2, let $TS_{\mathcal{X}_a}$ be the set of all possible approximated regular time series derived from the combination of different intervals of $\mathcal{X}_a$ by taking one interval from each time slot, i.e.

$$TS_{\mathcal{X}_a} = \{\langle I_{1,1}, I_{2,1}, \ldots, I_{n,1}\rangle, \ldots, \langle I_{1,l_1}, \ldots, I_{n,l_n}\rangle\}.$$

[2] Or minimum bounding hyper-rectangles in the d-dimensional case.

Let $X_a \in TS_{\mathcal{X}_a}$ and let $[l_{x_i}, u_{x_i}]$ be the interval of $X_a$ at time slot $i$. The distance $L_{L_p}(X_a, Y_a) = \sqrt[p]{\sum_{i=1}^{n} (\max\{0, \max\{l_{x_i}, l_{y_i}\} - \min\{u_{x_i}, u_{y_i}\}\})^p}$ is the smallest $L_p$-distance between all intervals of $X_a \in TS_{\mathcal{X}_a}$ and $Y_a \in TS_{\mathcal{Y}_a}$, whereas the distance $U_{L_p}(X_a, Y_a) = \sqrt[p]{\sum_{i=1}^{n} (\max\{u_{x_i} - l_{y_i}, u_{y_i} - l_{x_i}\})^p}$ is the largest $L_p$-distance between all intervals of $X_a \in TS_{\mathcal{X}_a}$ and $Y_a \in TS_{\mathcal{Y}_a}$. Aggregating these distance values by means of the distance function $\widetilde{dist}_p$, we obtain an interval of distances bound by $L_{dist}$ and $U_{dist}$. Now, we can lower bound each distance observation in $\widetilde{dist}_p(\mathcal{X}, \mathcal{Y})$ by
$LB_p(\mathcal{X}_a, \mathcal{Y}_a) = \{(L_{dist}(X_a, Y_a))^{|X_a| \cdot |Y_a|} | X_a \in TS_{\mathcal{X}_a}, Y_a \in TS_{\mathcal{Y}_a}\}$.
Analogously, we can upper bound each distance observation in $\widetilde{dist}_p(\mathcal{X}, \mathcal{Y})$ by
$UB_p(\mathcal{X}_a, \mathcal{Y}_a) = \{(U_{dist}(X_a, Y_a))^{|X_a| \cdot |Y_a|} | X_a \in TS_{\mathcal{X}_a}, Y_a \in TS_{\mathcal{Y}_a}\}$.

**Lemma 1.** *Let $X_a = \langle I_1^x, \ldots, I_n^x \rangle \in TS_{\mathcal{X}_a}$ and $\mathcal{Y}_a = \langle I_1^y, \ldots, I_n^y \rangle \in TS_{\mathcal{Y}_a}$ be approximated regular time series. For all $x = \langle x_1, \ldots, x_n \rangle$, $x_i \in I_i^x$ and for all $y = \langle y_1, \ldots, y_n \rangle$, $y_i \in I_i^y$, the following inequalities hold:*

$$L_{L_p}(\mathcal{X}_a, \mathcal{Y}_a) \leq L_p(x, y).$$

$$U_{L_p}(\mathcal{X}_a, \mathcal{Y}_a) \geq L_p(x, y).$$

A lower bound of the probability $\Pr(\widetilde{dist}_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon)$ can be defined as

$$\Pr_{LB}(\widetilde{dist}_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon) = \frac{|\{d \in UB_p(\mathcal{X}_a, \mathcal{Y}_a) | d \leq \varepsilon\}|}{s_{\mathcal{X}}^n \cdot s_{\mathcal{Y}}^n}$$

and an upper bound as

$$\Pr_{UB}(\widetilde{dist}_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon) = \frac{|\{d \in LB_p(\mathcal{X}_a, \mathcal{Y}_a) | d \leq \varepsilon\}|}{s_{\mathcal{X}}^n \cdot s_{\mathcal{Y}}^n}.$$

**Lemma 2.** *For any uncertain time series $\mathcal{X}$ and $\mathcal{Y}$, the following inequations hold:*

$$\text{(1)} \quad \Pr_{LB}(\widetilde{dist}_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon) \leq \Pr(\widetilde{dist}_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon)$$

$$\text{(2)} \quad \Pr_{UB}(\widetilde{dist}_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon) \geq \Pr(\widetilde{dist}_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon)$$

The proofs of Lemma 1 and 2 can be found in [16], but are omitted here due to space limitations.

The two following query types are based on an iterative filter-refinement policy. A queue $Q_{Ref}$ is used to organize all uncertain time series sorted by descending upper bounding probabilities $\Pr_{UB}(\widetilde{dist}_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon)$ w.r.t. the query object $\mathcal{Q}$.

## 3.3 Probabilistic Bounded Range Queries (PBRQ)

In an iterative process we remove the first element $\mathcal{X}$ of the queue $Q_{ref}$, compute its lower and upper bounding probabilities $\Pr_{LB}(\widetilde{dist}_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon)$ and

$\Pr_{UB}(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon)$, and filter $\mathcal{X}$ according to these bounds. If $\Pr_{LB}(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon) \geq \tau$, then $\mathcal{X}$ is a true hit and is added to the result set. If $\Pr_{UB}(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon) < \tau$, then $\mathcal{X}$ is a true drop and can be pruned. Otherwise, $\mathcal{X}$ has to be refined. Let us note that we do not immediately refine the object completely. Rather, the refinement is performed in several steps (1st level to 2nd level, 2nd level to exact representation). Details on the strategies for the step-wise refinement are presented below in Section 3.5. After the partial refinement step, $\mathcal{X}$ is again inserted into $Q_{Ref}$ if it cannot be pruned or reported as true hit according to the above conditions and is not refined completely yet. If an object $\mathcal{X}$ is refined completely, then obviously $\Pr_{LB}(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon) = \Pr_{UB}(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon) = \Pr(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon)$. The iteration loop stops if $Q_{Ref}$ is empty, i.e. all objects are pruned, identified as true hits before complete refinement, or are completely refined.

### 3.4 Probabilistic Ranking Range Query (PRRQ)

After initialization, the method *getNext()* can be called, returning the next object in the ranking. Obviously, an object $\mathcal{X}$ is the object with the highest probability if for all objects $\mathcal{Y} \in \mathcal{D}$ the following property holds: $\Pr_{LB}(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon) \geq \Pr_{UB}(\widetilde{dist}_p(\mathcal{Q},\mathcal{Y}) \leq \varepsilon)$. Since the candidate objects of the database are ordered by descending upper bounding probabilities in $Q_{Rank}$, we only need to check if the lower bounding probability of the first element in $Q_{Rank}$ is greater or equal to the upper bounding probability of the second element. If this test returns true, we can report the first object as the next ranked object. Otherwise, we have to refine the first object in $Q_{Rank}$ in order to obtain better probability bounds. As discussed above, this refinement is step-wise, i.e. several refinement steps are necessary in order to obtain the exact probability. The idea of the method *getNext()* is to iteratively refine the first object in $Q_{Rank}$ as long as the lower bounding probability of this element is lower than the upper bounding probability of the second element in $Q_{Rank}$.

### 3.5 Step-Wise Refinement of Probability Estimations

The aim for each refinement step is to be able to identify an uncertain time series as true hit or true drop. This aim is reached for an uncertain time series $\mathcal{X}$ if the probability interval $[\Pr_{LB}(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon), \Pr_{UB}(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon)]$ is above or below $\tau$. For this reason, we try to increase the lower bound of the probability $\Pr_{LB}(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon)$ in the case that

$$\tau - \Pr_{LB}(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon) \leq \Pr_{UB}(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon) - \tau$$

holds. Otherwise, we try to decrease $\Pr_{UB}(\widetilde{dist}_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon)$.

For increasing the lower bounds of the probabilities for $\mathcal{X}$ we have to refine those intervals which are intersected by the $\varepsilon$ value such that we refine first that

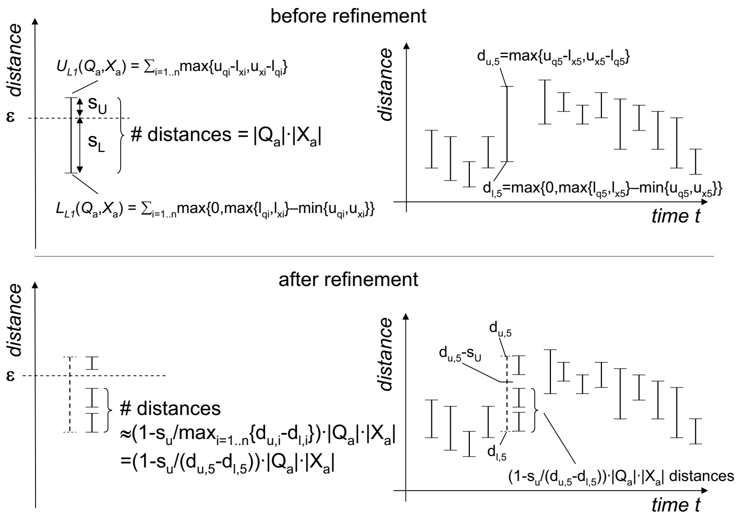

before refinement
distance
ε
U_L1(Q_a,X_a) = Σ_i=1..n max{u_qi-l_xi,u_xi-l_qi}
s_U
s_L
# distances = |Q_a|·|X_a|
L_L1(Q_a,X_a) = Σ_i=1..n max{0,max{l_qi,l_xi}-min{u_qi,u_xi}}
d_u,5=max{u_q5-l_x5,u_x5-l_q5}
d_l,5=max{0,max{l_q5,l_x5}-min{u_q5,u_x5}}
time t
after refinement
# distances ≈(1-s_u/max_i=1..n{d_u,i-d_l,i})·|Q_a|·|X_a| =(1-s_u/(d_u,5-d_l,5))·|Q_a|·|X_a|
d_u,5
d_u,5-s_U
d_l,5
(1-s_u/(d_u,5-d_l,5))·|Q_a|·|X_a| distances

**Fig. 2.** Refinement heuristics

approximated distance which probably will be resolved into a set of approximated distances that are clearly below $\varepsilon$ and approximates as many distances $d \in \widetilde{dist}_p(\mathcal{Q}, \mathcal{X})$ as possible. Here we use the following heuristic: The increase of the number of detected distances $d \in \widetilde{dist}_p(\mathcal{Q}, \mathcal{X})$ that are clearly below $\varepsilon$ can be estimated by

$$\widetilde{w} = (1 - \frac{s_u}{\max_{i=1..n}\{d_{u,i} - d_{l,i}\}}) \cdot |X_a| \cdot |Q_a|,$$

where $s_u = U_{dist}(Q_a, X_a) - \varepsilon$, $d_{u,i} = \max\{u_{q_i} - l_{x_i}, u_{x_i} - l_{q_i}\}$, $d_{l,i} = \max\{0, \max\{l_{q_i}, l_{x_i}\} - \max\{u_{q_i}, u_{x_i}\}\}$ and $|X_a| \cdot |Q_a|$ corresponds to the number of distances which are approximated by $U_{dist}(Q_a, X_a)$ and $L_{dist}(Q_a, X_a)$. The example depicted in Figure 2 shows the situation of the approximated distance $\widetilde{d} = (L_{L_1}(Q_a, X_a), U_{L_1}(Q_a, X_a))$ before (top) and after (bottom) the refinement step. The approximated distance $\widetilde{d}$ is refined by refining exactly one of the $n$ distance intervals in the time domain that correspond to $\widetilde{d}$. Obviously, the number of distances approximated by $\widetilde{d}$ is the product of the number $|Q_a|$ of regular time series approximated by $Q_a$ and the number $|X_a|$ of regular time series approximated by $X_a$. In order to estimate the number of approximated distances that fall below $\varepsilon$ after refining $\widetilde{d}$, we have to look at the distance intervals in the time domain. When refining a distance interval in the time domain, e.g. $(d_{l,5}, d_{u,5})$ in our example, then all resulting distance intervals that are clearly below $d_{u,i} - s_u$ correspond to the resulting approximated distances that are below $\varepsilon$. Since $\widetilde{w}$ has to be maximized, we should refine $\widetilde{d}$ by refining the largest time interval in the time domain. Finally, based on the described estimation, we refine the approximated distance for which $\widetilde{w}$ is maximal. In the case we want to decrease the upper bound of the probability $\Pr_{UB}(\widetilde{dist}_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon)$ we can use a very similar refinement strategy.

## 4 Summary of Experimental Results

In this short proposal, we just want to give a brief summary of our experimental results due to limited space. For the interesting reader we refer to [16] where a broader discussion of our experiments can be found. Our datasets are based on several artificial and real-world benchmark datasets derived from a wide range, including *CBF*, *GunX*, *SynCtrl* and *Leaf* from the UCI Time Series Data Mining Archive[3]. The time series are modified to get uncertain time series by means of sampling around the given exact time series values according to specific distribution functions (e.g. uniform and Gaussian). As discussed above, the computation of probabilistic similarity queries is CPU-bounded. To achieve a fair comparison which is independent of the implementation, we measured the efficiency by the average number of required calculations required to execute a query.

At first we measured the speed-up factor our approach yields compared to the straightforward approach naively computed as defined in Section 2. In the first experiment, we examine how our approach can speed up PBRQ and PRRQ for different datasets and varying sample rates. The speed-up factor of both query types is between $10^{75}$ and $10^{300}$ and increases exponentially with linearly increasing the sample rate. The rational for this is that the number of possible time-series instances increases exponentially with the time series length and the number of samples used for each time slot. Furthermore, we could show that our approach scales significantly better than the competitor w.r.t. the database size. Finally, we could experimentally show that our refinement strategy clearly outperforms more simple refinement strategies and that this superiority of our approach is robust w.r.t. all query parameters.

## 5 Conclusions

To the best of our knowledge, we propose the first approach for performing probabilistic similarity search on uncertain time series in this paper. In particular, we formalize the notion of uncertain time series and introduce two novel probabilistic query types for uncertain time series. Furthermore, we propose an original method for efficiently supporting these probabilistic queries using a filter-refinement query processing.

## References

1. Benjelloun, O., Sarma, A.D., Halevy, A., Widom, J.: ULDBs: Databases with Uncertainty and Lineage. In: Proc. 32nd Int. Conf. on Very Large Data Bases (VLDB 2006), Seoul, Korea, pp. 1249–1264 (2006)

[3] http://kdd.ics.uci.edu/

2. Re, C., Dalvi, N., Suciu, D.: Efficient top-k query evaluation on probalistic databases. In: Proc. 23rd Int. Conf. on Data Engineering (ICDE 2007), Istanbul, Turkey (2007)
3. Sen, P., Deshpande, A.: Representing and querying correlated tuples in probabilistic databases. In: Proc. 23rd Int. Conf. on Data Engineering (ICDE 2007), Istanbul, Turkey (2007)
4. Antova, L., Jansen, T., Koch, C., Olteanu, D.: Fast and Simple Relational Processing of Uncertain Data. In: Proc. 24th Int. Conf. on Data Engineering (ICDE 2008), Cancún, México (2008)
5. Cheng, R., Xia, Y., Prabhakar, S., Shah, R., Vitter, J.: Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data. In: Proc. 30th Int. Conf. on Very Large Data Bases (VLDB 2004), Toronto, Cananda, pp. 876–887 (2004)
6. Kriegel, H.P., Kunath, P., Pfeifle, M., Renz, M.: Probabilistic Similarity Join on Uncertain Data. In: Proc. 11th Int. Conf. on Database Systems for Advanced Applications, Singapore, pp. 295–309 (2006)
7. Cheng, R., Kalashnikov, D., Prabhakar, S.: Evaluating Probabilistic Queries over Imprecise Data. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD 2003), San Diego, CA, USA, pp. 551–562 (2003)
8. Kriegel, H.P., Kunath, P., Renz, M.: Probabilistic Nearest-Neighbor Query on Uncertain Objects. In: Kotagiri, R., Radha Krishna, P., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 337–348. Springer, Heidelberg (2007)
9. Lian, X., Chen, L.: Probabilistic ranked queries in uncertain databases. In: EDBT 2008, 11th International Conference on Extending Database Technology. Proceedings, Nantes, France, March 25-29, pp. 511–522 (2008)
10. Yi, K., Li, F., Kollios, G., Srivastava, D.: Efficient Processing of Top-k Queries in Uncertain Databases with x-Relations. IEEE Trans. Knowl. Data Eng. 20(12), 1669–1682 (2008)
11. Böhm, C., Pryakhin, A., Schubert, M.: Probabilistic Ranking Queries on Gaussians. In: Proc. 18th Int. Conf. on Scientific and Statistical Database Management (SSDBM 2006), Vienna, Austria, pp. 169–178 (2006)
12. Cormode, G., Li, F., Yi, K.: Semantics of Ranking Queries for Probabilistic Data and Expected Results. In: Proc. 25th Int. Conf. on Data Engineering (ICDE 2009), Shanghai, China, pp. 305–316 (2009)
13. Tao, Y., Cheng, R., Xiao, X., Ngai, W., Kao, B., Prabhakar, S.: Indexing Multi-Dimensional Uncertain Data with Arbitrary Probability Density Functions. In: Proc. 31th Int. Conf. on Very Large Data Bases (VLDB 2005), Trondheim, Norway, pp. 922–933 (2005)
14. Soliman, M., Ilyas, I.: Ranking with Uncertain Scores. In: Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, Shanghai, China, March 29-April 2, 2009, pp. 317–328 (2009)
15. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: Proc. 5-th Berkeley Symposium on Mathematical Statistics and Probability (1967)
16. Assfalg, J., Kriegel, H.P., Kröger, P., Renz, M.: Probabilistic Similarity Search for Uncertain Time Series. Tech. Rep. (2009), `http://www.dbs.ifi.lmu.de/~renz/technicalReports/uncertainTimeSeries.pdf`

# Reverse k-Nearest Neighbor Search Based on Aggregate Point Access Methods

Hans-Peter Kriegel, Peer Kröger, Matthias Renz, Andreas Züfle, and Alexander Katzdobler

Institute for Computer Science
Ludwig-Maximilians-University of Munich
{kriegel,kroegerp,renz,zuefle,katzdobl}@dbs.ifi.lmu.de

**Abstract.** We propose an original solution for the general reverse $k$-nearest neighbor (R$k$NN) search problem in Euclidean spaces. Compared to the limitations of existing methods for the R$k$NN search, our approach works on top of Multi-Resolution Aggregate (MRA) versions of any index structures for multi-dimensional feature spaces where each non-leaf node is additionally associated with aggregate information like the sum of all leaf-entries indexed by that node. Our solution outperforms the state-of-the-art R$k$NN algorithms in terms of query execution times because it exploits advanced strategies for pruning index entries.

## 1 Introduction

For a given query object $q$, a reverse $k$-nearest neighbor (R$k$NN) query returns all objects of a database that have $q$ among their actual $k$-nearest neighbors. In this paper, we focus on the traditional reverse $k$-nearest neighbor problem in feature databases and do not consider recent approaches for related or specialized R$k$NN tasks such as metric databases, the bichromatic case, mobile objects, etc. R$k$NN queries are important in many applications since the reverse $k$-nearest neighbors of a point $p$ reflect the set of those points that are influenced by $p$. As a consequence, a considerable amount of new methods have been developed that usually extend existing index structures for R$k$NN search. The use of an index structure is mandatory in a database context because R$k$NN query processing algorithms are — like all similarity query processing algorithms — I/O-bound. The naíve solution for answering R$k$NN queries would compute for all objects of the database the $k$-nearest neighbors ($k$NN) and report those objects that have the query object on their $k$NN list. In order to present efficient solutions for R$k$NN search, most existing approaches make specific assumptions in order to design specialized index structures. Those assumptions include the necessity that the value of the query parameter $k$ is fixed beforehand or the dimensionality of the feature space is low ($\leq$ 3). So far, the only existing approach for R$k$NN search that uses traditional (non-specialized) index structures and does not rely on the afore mentioned assumptions is called TPL [1]. In fact, the TPL approach computes a set of candidate points which is a superset of the result set in a first filter round. These candidates are used to prune other index entries already in this filter round. In a second refinement round, the $k$NNs of the candidates are computed to generate the final result.

In this paper, we extend the TPL approach in two important aspects. First, we generalize the pruning strategy implemented by the TPL approach by considering also other *entries* rather than only considering other *objects*. While the TPL approach usually needs to access several leaf nodes of the index although they may not include any true hits in order to start pruning other entries, we can start the pruning earlier and can save unnecessary refinements, i.e. disk accesses. Second, we show how entries may be pruned by themselves, which is a completely new pruning strategy not yet explored by the TPL approach. For this "self-pruning", we use the concept of aggregated point access methods like the aR-Tree [2,3]. Furthermore, we show how both the enhanced and the new pruning strategies can be integrated into the original TPL algorithm by altering only a very limited number of steps. Because our novel R$k$NN search algorithm implements both the enhanced and the new pruning strategies, it is expected to prune more entries than the TPL approach, i.e. it produces less I/O overhead and reduces query execution times considerably.

The reminder of this paper is organized as follows. In Section 2 we formally define the R$k$NN problem and discuss recent approaches for solving this problem. Section 3 presents our novel R$k$NN query algorithm. Our new approach is experimentally evaluated and compared to the state-of-the-art approach using synthetic and real-world datasets in Section 4. Last but not least, Section 5 concludes the paper.

## 2 Survey

### 2.1 Problem Defintion

In the following, we assume that $\mathcal{D}$ is a database of $n$ feature vectors, $k \leq n$, and $dist$ is the Euclidean distance[1] on the points in $\mathcal{D}$. In addition, we assume that the points are indexed by any traditional aggregate point access method like the aR-Tree family [2,3].

The set of *$k$-nearest neighbors* of a point $q$ is the smallest set $NN_k(q) \subseteq \mathcal{D}$ that contains at least $k$ points from $\mathcal{D}$ such that

$$\forall o \in NN_k(q), \forall \hat{o} \in \mathcal{D} - NN_k(q) : dist(q, o) < dist(q, \hat{o}).$$

The point $p \in NN_k(q)$ with the highest distance to $q$ is called the *$k$-nearest neighbor* ($k$NN) of $q$. The distance $dist(q, p)$ is called *$k$-nearest neighbor* distance ($k$NN distance) of $q$, denoted by $nndist_k(q)$.

The set of *reverse $k$-nearest neighbors* (R$k$NN) of a point $q$ is then defined as

$$RNN_k(q) = \{p \in \mathcal{D} \mid q \in NN_k(p)\}.$$

The naive solution to compute the R$k$NN of a query point $q$ is rather expensive. For each point $p \in \mathcal{D}$, the $k$NN of $p$ is computed. If the distance between $p$ and $q$ is smaller or equal to the $k$NN distance of $p$, i.e. $dist(p, q) \leq nndist_k(q)$, then $q \in NN_k(p)$ which in turn means that point $p$ is a R$k$NN of $q$, i.e. $p \in RNN_k(q)$. The runtime complexity of answering one R$k$NN query is $O(n^2)$ because for all $n$ points, a $k$NN query needs to be launched which requires $O(n)$ when evaluated by a sequential scan. The costs of an R$k$NN query can be reduced to an average of $O(n \log n)$ if an index such as the R-Tree [4] or the R*-Tree [5]) is used to speed-up the $k$NN queries.

[1] Let us note that the concepts described here can also be extended to any $L_p$-norm.

## 2.2 Related Work

Here, we focus on feature vectors rather than on metric data. Thus, we do not consider approaches for metric data [6,7,8,9] as competitors. Usually, these approaches are less efficient on Euclidean data because they cannot make use of the Euclidean geometry. Existing approaches for the Euclidean R$k$NN search can be classified as self-pruning approaches or mutual-pruning approaches.

**Self-pruning** approaches are usually designed ontop of a hierarchically organized tree-like index structure. They try to estimate the $k$NN distance of each index entry $E$, i.e. $E$ can be a database point or an intermediate index node. If the $k$NN distance of $E$ is smaller than the distance of $E$ to the query $q$, then $E$ can be pruned. Thereby, self-pruning approaches do usually not consider other points (database points or index nodes) in order to estimate the $k$NN distance of an entry $E$ but simply precompute $k$NN distances of database points and propagate these distances to higher level index nodes. The RNN-Tree [10] is an R-Tree-based index that precomputes for each point $p$ the distance to its 1NN, i.e. $nndist_1(p)$ and index for each point $p$ a sphere with radius $nndist_1(p)$ around $p$. The RdNN-Tree [11] extends the RNN-Tree by storing the points of the database itself in an R-Tree rather than circles around them. For each point $p$, the distance to $p$'s 1NN, i.e. $nndist_1(p)$, is aggregated. For each intermediate entry $E$, the maximum of the 1NN distances of all child entries is aggregated. Since the $k$NN distances need to be materialized, both approaches are limited to a fixed value of $k$ and cannot be generalized to answer R$k$NN-queries with arbitrary values of $k$. In addition, approaches based on precomputed distances can generally not be used when the database is updated frequently. Otherwise, for each insertion or deletion of points, the $k$NN distances of the points influenced by the updates need to be updated as well which is a considerably high computational overhead.

**Mutual-pruning approaches** use other points to prune a given index entry $E$. For that purpose, they use special geometric properties of the Euclidean space. In [12] a two-way filter approach for supporting R1NN queries is proposed that provides approximate solutions, i.e. may suffer from false alarms and incomplete results. A different approach is presented in [13] R$k$NN queries. Since it is based on a partition of the data space into equi-sized units where the border lines of the units are cut at the query point $q$ and the number of such units increases exponentially with the data dimensionality, this approach is only applicable for 2D data sets. In [1] an approach for R$k$NN search was presented, that can handle arbitrary values of $k$ and may be applied to arbitrary dimensional feature spaces. The method is called TPL and uses any hierarchical tree-based index structure such as an R-Tree to compute a nearest neighbor ranking of the query point $q$. The key idea is to iteratively construct Voronoi hyper-planes around $q$ w.r.t. to the points from the ranking. Points and index entries that are beyond $k$ Voronoi hyper-planes w.r.t. $q$ can be pruned and need not to be considered for Voronoi construction. The idea of this pruning is illustrated in Figure 1 for $k = 1$. Entry $E$ can be pruned, because it is beyond the Voronoi hyper-plane between $q$ and candidate $x$. To decide whether an entry $E$ can be pruned or not, TPL employs a special trimming function that examines if $E$ is beyond $k$ hyper-planes w.r.t. all current candidates. In addition, if $E$ cannot be pruned but one or more hyper-planes intersect the page region of $E$, the trimming function trims the hyper-rectangular page region of $E$ and, thus, potentially

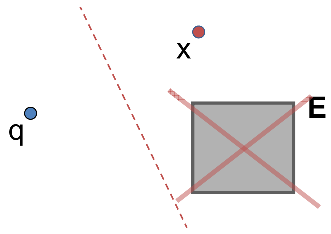


**Fig. 1.** TPL pruning ($k = 1$)

decreases the MinDist of $E$ to $q$. As a consequence of such a trimming, $E$ may move towards the end of the ranking queue when reinserted into this queue. This increases the chance that $E$ can be pruned at a later step, because until then new candidates have been added. The remaining candidate points must be refined, i.e. for each of these candidates, a $k$NN query must be launched.

## 3 RkNN Search Using Multiple Pruning Strategies

### 3.1 Combining Multiple Pruning Strategies

As discussed above, we want to explore self-pruning as well as mutual pruning possibilities in order to boost R$k$NN query execution. Our approach is based on an index structure $\mathcal{I}$ for point data which is based on the concept of minimal-bounding-rectangles, e.g. the R-tree family including the R-tree [4], the $R^*$-tree [5] and the X-tree [14]. In particular, we use multi-resolution aggregate versions of these indexes as described in [2,3] that e.g. aggregate for each index entry $E$ the number of objects that are stored in the subtree with root $E$. The set of objects managed in the subtree of an index entry $E \in \mathcal{I}$ is denoted by *subtree*$(E)$. Note that the entry $E$ can be an intermediate node in $\mathcal{I}$ or a point, i.e. an object in $\mathcal{D}$. In the case that the entry $E \in \mathcal{I}$ is an object (i.e. $E = e \in \mathcal{D}$) then *subtree*$(E) = \{e\}$. The basic idea of our approach is to apply the pruning strategy mentioned above during the traversal of the index, i.e. to identify true drops as early as possible in order to reduce the I/O cost by saving unnecessary page accesses. The ability to prune candidates already at the directory level of the index implies that a directory entry is used to prune itself (self-pruning) or other entries (mutual-pruning).

For an $RNN_k$ query with $k \geq 1$, an entry $E$ can be pruned by another entry $E'$ if there are at least $k$ objects $e' \in$ *subtree*$(E')$ such that $E$ is behind the Voronoi hyperplane between $q$ and $e'$, denoted by $\perp(q, e')$. In general, we call a hyperplane $\perp(q, e)$ *associated with* the object $e$. Note, that a hyperplane $\perp(q, e)$ represents all points in the object space having equal distances to $q$ and to $e$, i.e. for all points $p \in\perp(q, e)$ $dist(p, q) = dist(p, e)$ holds as shown in the example depicted in Figure 2. An object or point is called to be *behind* a hyperplane $\perp(q, e)$ if it is located within the half space determined by $\perp(q, e)$ which is opposite to the half space containing the query object $q$. Consequently, objects which are behind a hyperplane $\perp(q, e)$ are closer to $e$ than to $q$, i.e. object $o$ is behind $\perp(q, e)$ implies that $dist(o, q) > dist(o, e)$. Furthermore, an entry $E$ (intermediate index node) is called to be *behind* a hyper plane $\perp(q, e)$, if all points of the entire page region of $E$ are behind $\perp(q, e)$. In our example, object $x$ as well as entry $E$ are behind $\perp(q, e)$.

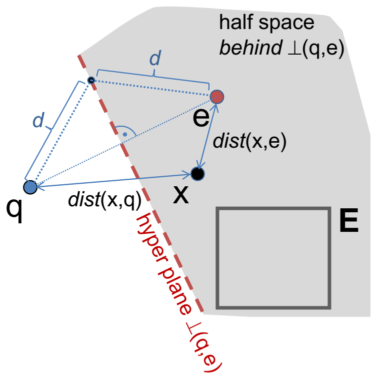


**Fig. 2.** Voronoi hyperplane between two objects $q$ and $e$ determining the half space which can be used to prune R$k$NN candidates

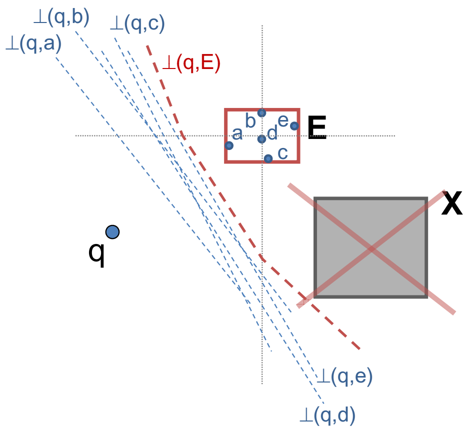


**Fig. 3.** Hyperplanes associated with all objects of an index entry $E$

The key idea of the directory-level-wise pruning is to identify a hyperplane $\perp(q, E)$ which can be associated with an index entry $E$ and which conservatively approximates the hyperplanes associated with all objects $e$ in the subtree of $E$, i.e. $e \in \mathit{subtree}(E)$. Figure 3 illustrates the idea of this concept. We say that the hyperplane associated with an index entry $E$ is *related to* the set of objects in the subtree of $E$. Since we assume that the number of objects stored in the subtree of an index entry $E$ is known, we also know for the hyperplane associated with that index entry $E$, $\perp(q, E)$, how many objects this hyperplane relates to. We can use this information in order to prune entries according to $E$ without accessing the child entries of $E$. For example, if the number of objects that relate to $\perp(q, E)$ is greater than the query parameter $k$, we can prune all points and entries that are behind $\perp(q, E)$. In Figure 3, entry $X$ can be pruned for $k \leq 5$ because $|\mathit{subtree}(E)| = 5$. As mentioned above, to obtain the number of objects stored in a subtree of an entry, we can exploit the indexing concept as proposed in [2]. This concept allows to store for each index entry $E$ the number of objects stored in

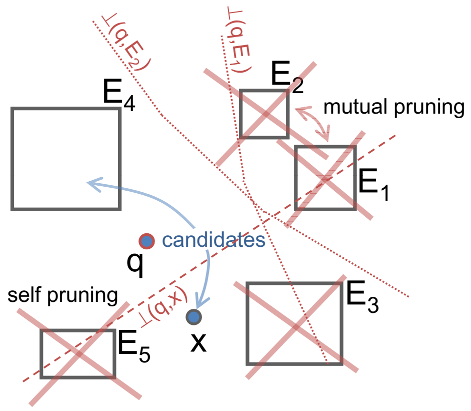


**Fig. 4.** Pruning potentials using different pruning strategies

the subtree that has $E$ as its root, i.e. the aggregate value $|\mathit{subtree}(E)|$ is stored along with each entry $E$. For example, the aggregate R-tree (aR-tree) [2,3] is an instance of this indexing concept. Then, the number of objects that are related to $\perp(q, E)$ equals $|\mathit{subtree}(E)|$.

In addition, we can use these considerations also for the self-pruning of entries. If an entry stores more than $k$ objects in its subtree, i.e. $|\mathit{subtree}(E)| > k$, and $E$ is behind the hyperplane that is associated with itself, $\perp(q, E)$, then $E$ can be pruned. The rational for this is that $|\mathit{subtree}(E)| > k$ objects relate to $\perp(q, E)$, i.e. more than $k$ hyperplanes are approximated by $\perp(q, E)$. As a consequence, each object $o \in \mathit{subtree}(E)$ is behind at least $k$ hyperplanes. This self-pruning can be performed without considering any other entry. For example, for $k \leq 4$, the entry $E$ in Figure 3 can also be pruned without considering any other entry because each point in the subtree of $E$ is behind the hyperplane of all four other points in $E$.

Figure 4 visualizes the benefits of using higher level mutual-pruning and self-pruning on a fictive 2D Euclidean database indexed by an R-Tree-like structure. The hyperplane associated with an index entry $E$ is denoted by $\perp(q, E)$. If we assume that each of the entries $E_1$, $E_2$, and $E_5$ stores more than $k$ objects in its particular subtree, all three entries can be pruned by the self-pruning strategy which does not consider any other entries. This can be done because all three entries are lying behind those hyperplanes that are associated with themselves. On the other hand, entries $E_1$, $E_2$ and $E_3$ can be pruned by the mutual-pruning strategy which is based on heuristics that consider other entries. While $E_3$ can only be pruned for $k = 1$ due to the hyperplane associated with object $x$, both $E_1$ and $E_2$ even can be pruned for $k \geq 1$ with the assumption that each of the values of $|\mathit{subtree}(E_1)|$ and $|\mathit{subtree}(E_2)|$ is greater than or equal to $k$. Let us note that a mutual-pruning approach like [1] needs at least one exact object to prune other entries, i.e. $E_1$, $E_2$ and $E_5$ can neither prune themselves nor prune each other. In that case, only entry $E_3$ could be pruned and all other entries need to be refined. The extension of the mutual-pruning strategy and the combination with the self-pruning strategy allows us to prune all candidates except for $E_4$ and object $x$. This simple example illustrates the potential benefit of our approach. In other words, the aim of our novel method is to

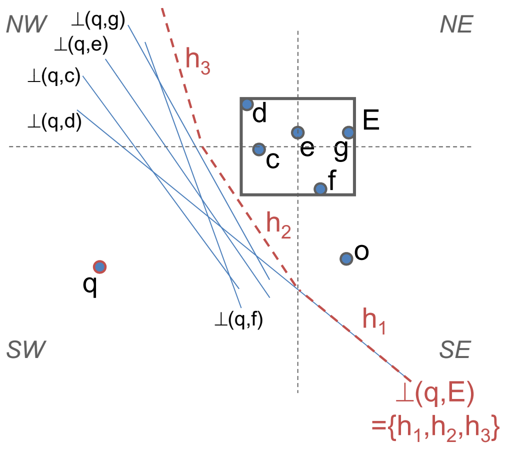


**Fig. 5.** Conservative approximation $\perp(q, E)$ of the hyperplanes associated with all objects of an index entry $E$

provide the advantages of the mutual-pruning and the self-pruning approaches by fading out the drawbacks of both, thus, providing the "best of two worlds". As a consequence our solution is expected to outperform the existing approach [1] in terms of query execution times because of the advanced pruning capabilities that are derived from the combination of the self-pruning and mutual-pruning potentials on higher index levels.

## 3.2 Intermediate Index Entry Hyperplanes

The most important question is, how to derive a hyperplane $\perp(q, E)$ associated with an entry $E \in \mathcal{I}$. This hyperplane $\perp(q, E)$ associated with an index entry $E$ is required to constitute a conservative approximation of the hyperplanes associated with all objects $o$ in the subtree of $E$, i.e. $o \in \mathit{subtree}(E)$. In fact, we will see that $\perp(q, E)$ is defined by means of a set of hyperplanes rather than by only one hyperplane, depending on the location in the feature space. In general, a set of hyperplanes $H$ is called *conservative approximation* of another set of hyperplanes $H'$, if all objects related to the hyperplane $h \in H$ are definitely behind each hyperplane $h' \in H'$, formally:

$$(\forall h \in H : o \text{ behind } h) \Rightarrow (\forall h' \in H' : o \text{ behind } h')$$

An example is shown in Figure 5, where the hyperplane $\perp(q, E)$ associated with the index entry $E$ forms a conservative approximation of all hyperplanes that are associated with the objects covered by $E$. The hyperplane approximation consists of the three hyperplanes $h_1$, $h_2$ and $h_3$. Objects that are behind these three hyperplanes, e.g. object $o$, are definitely behind all hyperplanes that are associated with the objects covered by $E$, independent of their location in $E$. Such an approximation is sensible if we assume that the set $H$ is much smaller than the set $H'$ and, thus, can be used to prune entries more efficiently.

In the following, we show how we can define such a set of hyperplanes $\perp(q, E)$ associated with an index entry $E$ which conservatively approximates the hyperplanes

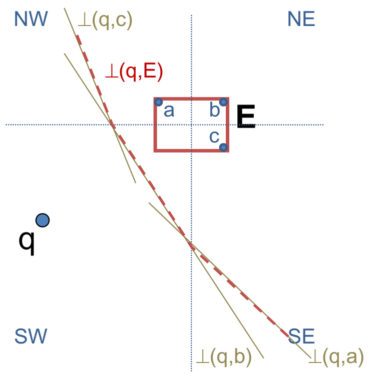


**Fig. 6.** Computation of conservative hyperplane approximations

of all objects stored in the subtree of $E$. As mentioned above, a hyperplane associated with an object $o$ represents all points $p$ which have the same distance to the query point $q$ and to $o$. In addition, we know that all objects stored in the subtree of an index entry $E$ are located inside the minimum bounding hyper-rectangle (mbr) that defines the page region of $E$. Thus, we can determine a conservative hyperplane representation of all points stored in the subtree of entry $E$ if we replace the distances between the hyperplane points $p \in \perp(q, E)$ and $o \in \mathit{subtree}(E)$ by the maximum distance between $p$ and the mbr-region of $E$. Figure 6 illustrates the computation of such a conservative approximation for a given index entry $E$ in a two-dimensional feature space. First, we have to specify the maximum distance between an mbr-region of the index entry $E$ and any point in the vector space. It suffices to find for each point $p$ in the vector space the point $e$ within the mbr-region which has the maximum distance to $p$. This can be done by considering partitions of the vector space which are constructed as follows: in each dimension the space is split paraxially at the center of the mbr-region. As illustrated for the two-dimensional example in Figure 6, we obtain partitions denoted by $NW$, $NE$, $SE$ and $SW$. In each of these partitions $P$, the corner point of the mbr-region which lies within the diagonally opposite partition is the mbr-region point which has the maximum distance to all points within $P$. In our example, for any point $p$ in $SW$ the maximum distance of $p$ to $E$ is the distance between $p$ and point $b$ in partition $NE$. Consequently, the hyperplane $\perp (q, b)$ is a conservative approximation of all hyperplanes between points within the mbr-region of $E$ and the points within the partition $SW$. This way, in our example the hyperplane associated with $E$ is composed of the three hyperplanes $\perp (q, a)$, $\perp (q, b)$ and $\perp (q, c)$. Generally, the conservative approximation of an mbr-region in a $d$-dimensional space consists of at most $2^d$ hyperplanes. This is due to the fact that a $d$-dimensional space can be partitioned into $2^d$ partitions according to an mbr-region, such that the maximums distance between each point $p$ in a partition and an mbr-region $E$ is defined by exactly one point within $E$.

In the following we will show that if we construct the hyperplane approximation as mentioned above we achieve in fact a conservative hyperplane approximation. The example illustrated in Figure 7 visualizes the scenario described in the following lemma.

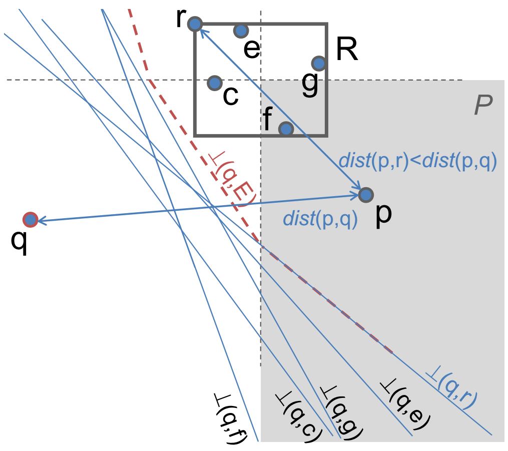


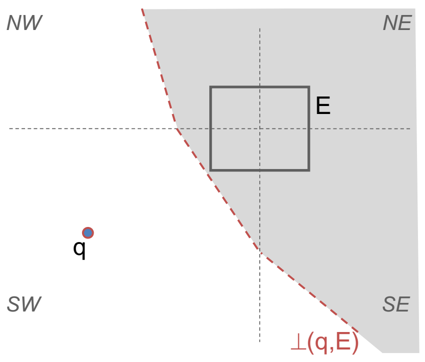
**Fig. 7.** Conservative approximation $\perp(q, E)$ of the hyperplanes associated with all objects of an index entry $E$

**Lemma 1.** *Let $q$ be a (query) point, $R$ be an mbr-region in a $d$-dimensional space and $p$ be any $d$-dimensional point. Furthermore, let $P$ denote the space partition which is generated by splitting the space paraxially at the center of the mbr-region $R$ in each dimension and let $P$ be the partition containing the point $p$. Then, the hyperplane between $q$ and the corner point $r$ of $R$ which lies within the diagonally opposite partition builds a conservative approximation of all hyperplanes between $q$ and all other points in $R$ within the partition $P$. In other words, all points in $P$ that are behind $\perp(q, r)$ are also behind each hyperplane between $q$ and any other point in $R$.*

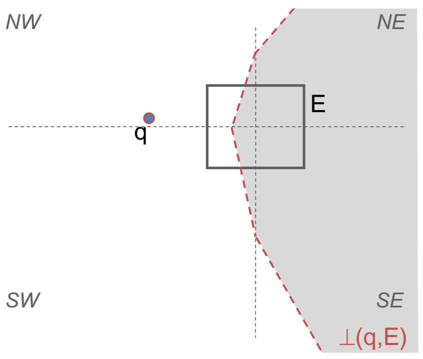
*Proof.* By definition, each point $p$ behind the hyperplane $\perp(q, r)$ has a smaller distance to the point $r$ than to $q$, i.e. $dist(p, r) < dist(p, q)$. Furthermore, $r$ is assumed to be the point in $R$ with the maximal distance to $p$, i.e. the distance from $p$ to any point $p'$ in $R$ is smaller or equal to $dist(p, r)$. Consequently, the distance between $p$ and $p'$ is smaller than the distance between $p$ and $q$. Since the hyperplane $\perp(q, p')$ associated with any point $p'$ in $R$ only contains points having equal distance to $p'$ and $q$ by definition, $\perp(q, p')$ cannot contain such a point $p$ which is assumed to be behind $\perp(q, r)$. As a consequence, no hyperplane associated with $q$ and any point in $R$ is behind $\perp(q, r)$ within the region $P$.

According to Lemma 1, we can combine all hyperplane approximations of all regions associated with an mbr-region $R$ into a set of hyperplanes that conservatively approximate the hyperplanes of all points in $R$ w.r.t. the entire data space. The two-dimensional example illustrated in Figure 5 shows that the combination of the three hyperplanes, marked by the red dotted poly-line, conservatively approximates the hyperplanes associated with all points within $E$. Each index entry $X$ which lies behind the hyperplane approximation $\perp(q, E)$ associated with the entry $E$ also lies behind each of the hyperplanes associated with each object in $E$.

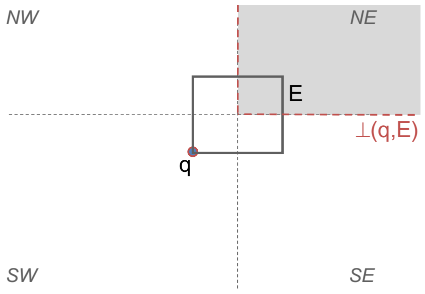
Note, that the shape of the hyperplane associated with an mbr-region depends on the topology between the query point $q$ and the mbr-region. Figure 8 exemplarily shows



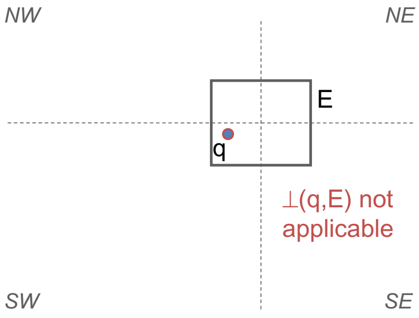
(a) Case 1.



(b) Case 2.



(c) Case 3.



(d) Case 4.

**Fig. 8.** Conservative hyperplane approximations associated with point $q$ and mbr-region $E$ for different cases w.r.t. the topology of $q$ and $E$

four cases with different topologies in the two-dimensional space. Case 1 (cf. Figure 8(a)) shows the standard case where the hyperplane associated with $E$ can be represented by three regular hyperplanes, i.e. three hyperplanes each associated with a single point. As shown in case 2 (cf. Figure 8(b)), the hyperplane associated with $E$ is generally represented by four regular hyperplanes. Case 3 (cf. Figure 8(c)) shows the special scenario where $q$ is located at one of the corner points of the mbr-region. Here, the hyperplane associated with $E$ is represented by only two regular hyperplanes. An interesting case is case 4 where none of the four regular hyperplanes of which the hyperplane associated with $E$ is constructed are applicable. The reason is that none of the four regular hyperplanes which are used to construct the hyperplane associated with $E$ intersects the region it relates to and, thus, cannot be used to prune any candidate.

## 3.3 Pruning Candidates

In the following, we show how the hyperplanes $\perp(q, E)$ associated with an index entry $E$ can be used to prune itself or other entries. Here, we assume that an R$k$NN-query with $k \geq 1$ is issued. Since, the hyperplanes $\perp(q, E)$ associated with an index entry $E$ approximate all hyperplanes associated with all objects within the subtree of $E$, any index entry $X$ which lies behind $\perp(q, E)$ in fact must lie at least behind $|subtree(E)|$ hyperplanes, and, thus can be pruned by $E$ if $|subtree(E)| \geq k$. For this reason, we assign a weight $w(\perp(q, E)) \in \mathbb{N}^+$ to each hyperplane $\perp(q, E)$ associated with an index entry $E$. The weight $w(\perp(q, E))$ denotes the number of hyperplanes which are approximated by $\perp(q, E)$. Since, we use aggregate index structures, we assume that the number of objects managed by an index entry $E$ is accessible without the need to refine the entry $E$. Once, the hyperplanes $\perp(q, E)$ for an entry $E$ are built, we can assign the weight $w(\perp(q, E)) = subtree(E)$ to them. Obviously, a hyperplane associated with an object has the weight 1.

In summary, if we assume that we already determined a set of $n$ hyperplanes $\mathcal{S} = \{\perp(q, E_1), \perp(q, E_1), \cdots, \perp(q, E_n)\}$ behind which an index entry $E$ lies, then, we can prune $E$, if $n \geq k$.

### 3.4 The R*k*NN Search Algorithm

The benefit of our formalization presented above is that the two pruning strategies, general mutual-pruning and self-pruning, can now easily be integrated into the TPL algorithm. In other words, our concepts allow us to use the TPL algorithm as a framework for R$k$NN search. Thus, our novel algorithm also relies on a filter step and a refinement step. Our filter step is very similar to the filter step of the TPL approach. We also manage a heap $H$ to compute a nearest neighbor ranking, a set of candidates points $S_{cnd}$ and a set of pruned entries $S_{rfn}$. The key difference is that we call the trimming function in a different way. Instead of trimming an index entry or a database point w.r.t. the candidate points in $S_{cnd}$, we use all entries/points in $H$, $S_{rfn}$, $S_{cnd}$ for trimming. This implements the advanced mutual-pruning already on the directory level of the index as well as the self-pruning of index entries. In addition, we have to generalize the trimming function such that the clipping of page regions considers the weight of each hyperplane. The clipping algorithm sketched in [1] can easily be adapted for this purpose. Finally, our refinement step is algorithmically also very similar to the refinement step of TPL. However, it is expected that it requires less disc accesses because usually less candidates need to be refined. Intuitively, the refinement step tests for each point in $S_{cnd}$ if the query $q$ is among its $k$NN list by considering and iteratively refining the points and index entries in $S_{rfn}$.

## 4 Experimental Evaluation

We compared our novel approach for R$k$NN search, hereafter referred to as HPKRNN (short for Hyper-Plane based R$k$NN), with TPL [1] the current state-of-the-art algorithm. All experiments are based on an R*-Tree with a page sizes of 32 Byte and 1KByte. For all experiments, we executed 100 sample R$k$NN queries and averaged the results.

Our experiments are conducted on four synthetic data sets with different features that are summarized in Table 1.

**Table 1.** Features of the synthetic data sets used for evaluation

| Name | size | dimension | distribution |
|---|---|---|---|
| Synth1 | 3,500 | 2 | uniform |
| Synth2 | 3,500 | 2 | 6 Gaussians |
| Synth3 | 1,000,000 | 2 | 6 Gaussians |
| Synth4 | 1,500 | 20 | uniform |

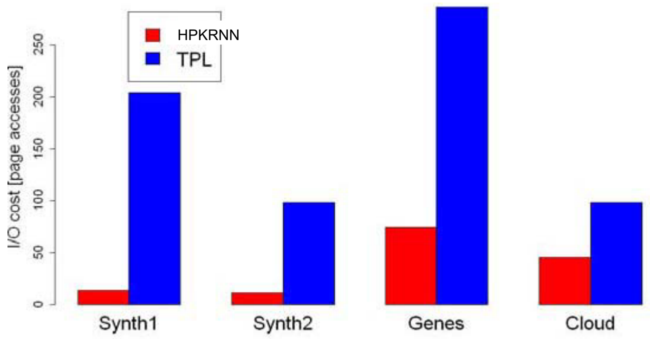
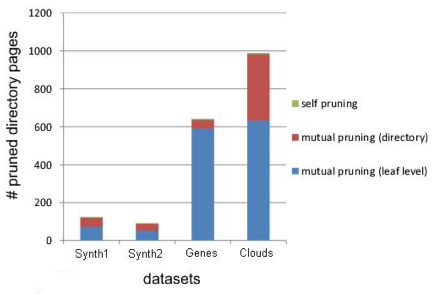


**Fig. 9.** Comparison of HPKRNN and TPL processing R1NN queries

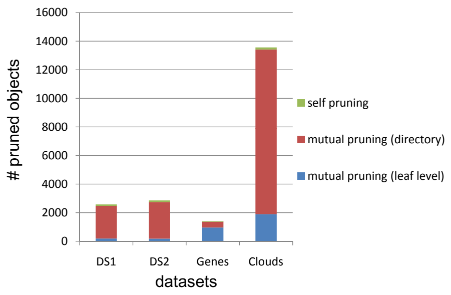
Additionally, we conducted experiments on two real-word data sets. The "Genes" data set contains appr. 5,000 points in a 5D space representing the expression levels of genes. The data set "Cloud" contains 9D weather parameters recorded at appr. 17,100 different locations in Germany.

## 4.1 Evaluation of the I/O-Cost

Figure 9 displays the performance of the competitors on four data sets when processing R1NN queries using an R*-tree with a page size of 32 byte. We used Synt1 and Synt2 because they feature different characteristics, as well as the two real-world data sets Genes and Cloud. It can be seen that our novel HPKRNN algorithm significantly



(a) Pruning power w.r.t. directory nodes.



(b) Pruning power w.r.t. data objects.

**Fig. 10.** Benefit of different pruning strategies for HPKRNN

outperforms the TPL approach in terms of I/O costs and, thus, query execution times. The reason for this clear performance boost over the mutual-pruning approach TPL can be derived from Figure 10 where the number of self-pruned objects, the number of mutual-pruned objects on the leaf level, and the number of mutual-pruned objects on higher levels in the index are displayed separately for our HPKRNN approach. As it can be observed from this figure, the combination of the pruning strategies on multiple levels as performed by HPKRNN is beneficial and superior over using only mutual-pruning on the leaf level of the index as it is done by TPL. Especially when considering pruned objects, the big positive effect of the mutual-pruning at the directory level becomes obvious (cf. Figure 10(b)). But also for pruning directory pages, especially the mutual-pruning at the directory level erases a large number of candidates (cf. Figure 10(a)). It can also be observed from both charts in Figure 10, that contribution of the self-pruning strategy seems to be less important on the applied data sets.

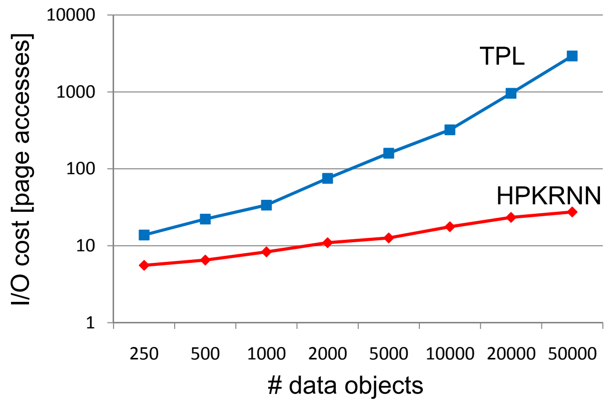


**Fig. 11.** Scalability of the competitors w.r.t. the data set size using an R*-tree with a page size of 32 Byte

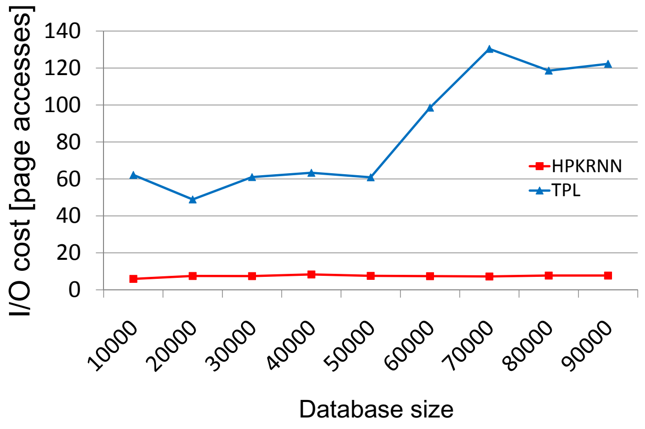
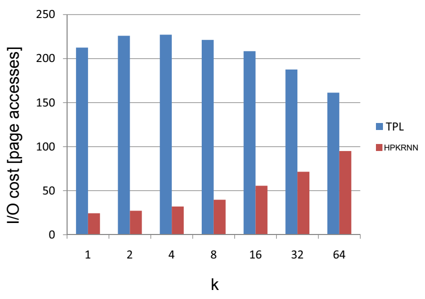
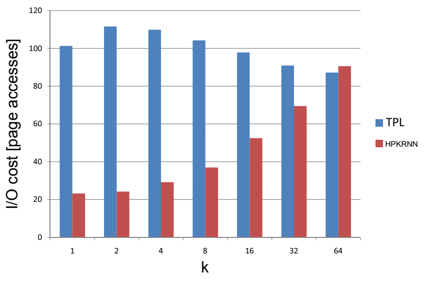


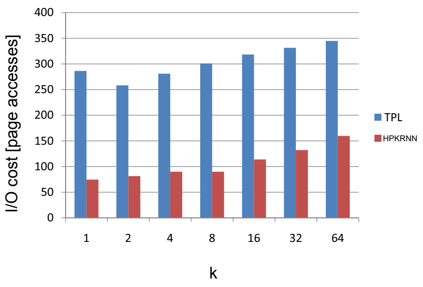
**Fig. 12.** Scalability of the competitors w.r.t. the data set size using an R*-tree with a page size of 1024



(a) Synth1.



(b) Synth2.



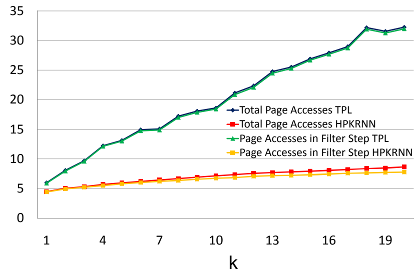
(c) Gene.

**Fig. 13.** Performance of the competitors in the number of pages accessed w.r.t. different values of $k$ using an R*-Tree with a pagesize of 32 Byte

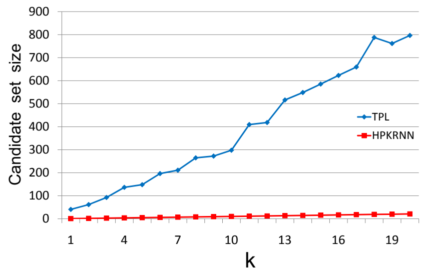
Next, we evaluated the scalability of the competitors w.r.t. the number of data objects $n$. Figure 11 displays the results for an R*-Tree with a pagesize of 32Byte and Figure 12 reports the results for an R*-Tree with a pagesize of 1kB. Again, the performance gain of our HPKRNN algorithm over the TPL method remains significant with varying number of data objects. In particular for large databases our method outperforms the TPL method by up to two orders of magnitude.

In the next experiments, we evaluated the impact of the query parameter $k$ on the scalability of the competitors. The resulting performances are visualized in Figure 13. Again, our method clearly outperforms the TPL approach on most data sets especially for smaller values of $k$. With increasing $k$, the gap between both approaches decreases. A reason for this might be that the self-pruning and the mutual-pruning at the directory level becomes less selective in this case. Rather, with increasing $k$, most directory pages and objects are pruned with the mutual-pruning at the leaf level. However, as observable from the Synt1 and Gene data sets, this effect is only visible for rather high values of $k$.

Figure 14 shows the influence of the query parameter $k$ when using an R*-Tree with a larger page-size (in this case 1K). Here, we evaluated the number of page accesses (cf. Figure 14(a)) and the maximum size of the candidate set (cf. 14(b)) produced by each method. The relatively large number of page accesses of the TPL algorithm



(a) Number of page accesses



(b) Size of the candidate set

**Fig. 14.** Performance of the competitors w.r.t. different values of $k$ using an R*-Tree with a page-size of 1024 Byte

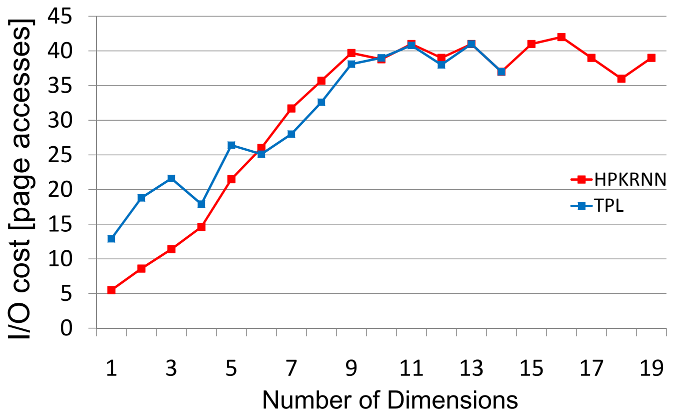


**Fig. 15.** Performance of the competitors w.r.t. the number of dimensions $d$ of the data set

(Figure 14(a)) can be explained by a rapidly growing number of entries in the candidate set of the TPL algorithm, shown in Figure 14(b). Additionally, it can be observed in Figure 14(a) that the vast majority of page accesses in both approaches occur in the respective filter steps whereas the refinement round requires only a small number of page accesses. This is an interesting observation because HPKRNN is clearly superior to TPL in the filter step (cf. Figure 14(a)) and additionally in the number of objects that need to be refined (cf. Figure 14(b)), i.e. produces a considerably smaller number of candidates with a significantly smaller amount of page accesses.

We also evaluated the scalability of our approach w.r.t. the number of dimensions $d$ of the data set using the 20-dimensional data set “Synth4” containing 1,500 data points. Figure 15 shows the results of the experiment in which we subsequently increased the number of relevant dimensions. In this experiment, we chose a fixed capacity of data points that can be stored in an R*-Tree node to keep the results comparable, in particular, a capacity of 30 data points for directory nodes and a capacity of 60 data points for data (leaf) nodes.

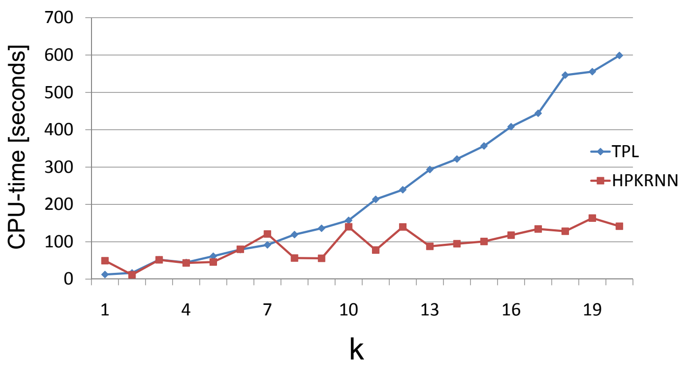


**Fig. 16.** Scalability of the competitors in terms of CPU costs w.r.t. $k$

It can be observed that our HPKRNN algorithm outperforms the TPL method for dimensions less or equal to five, i.e. $d \leq 5$. For higher dimensions, both approaches appear to perform very similar. This can be explained by the general bad performance of R*-Trees on more than 5-dimensional data. In order for an minimal bounding rectangle (mbr) to contain its minimal number of entries, it has to cover an increasingly large fraction of space in each dimension.

## 4.2 Evaluation of the CPU-Cost

Next, we evaluated the time required to compute the results of R$k$NN-queries with respect to the database size in terms of CPU-time. We also compared the CPU costs of our HPKRNN approach to the CPU costs of the TPL approach. Here, we only compared the time required for the refinement step, because, in [1], the CPU costs of the filter step is boosted using heuristics based on Hilbert values. Since our TPL algorithm does not implement this heuristic, we decided to omit experiments on the runtime of the filter step of the TPL, in order to avoid unfair comparisons. Note that not using these heuristics proposed in the TPL approach does not affect the I/O costs of the TPL approach. The result of this experiment is shown in Figure 16 for different values of $k$. It can be seen that our method is competetive with the TPL approach in terms of CPU runtime. This indicates that, since we need more effort to compute hyper-planes between the query and a directory node, a less number of hyper-planes is needed to prune objects and nodes. This coincides directly with the observation made above that our HPKRNN method produces less candidates in the filter step because the number of hyper-planes computed is determined by the number of candidates we have during the filter step.

## 4.3 Summary

In summary, the conducted experiments confirm that our novel approach clearly outperforms the current state-of-the-art approach because it combines multiple pruning strategies rather than implementing only one pruning paradigm. Even though our novel pruning strategies may produce an additional overhead at the CPU end, we also showed that the CPU costs of our algorithm is competitive with the CPU costs of the existing method. Furthermore, our approach saves a considerably number of page accesses during the filter step and the refinement step. As a consequence, since the R$k$NN problem is I/O-bound for large data sets, our new algorithm needs significantly less time to report the results of R$k$NN queries than the current state-of-the-art approach for this problem.

## 5 Conclusions

In this paper, we propose a generalization of the TPL algorithm which is the current state-of-the-art approach to Euclidean R$k$NN search. Our solution extends the TPL method in two important ways. First, the mutual-pruning strategy of TPL is generalized so that it can be applied already on higher levels of the index. Second, we introduced a new pruning paradigm called self-pruning. The generalization of the mutual-pruning strategy and its combination with the new self-pruning strategy helps to explore the full pruning potentials in order to reduce query execution times. Our experimental evaluation confirms that our new solution outperforms the existing methods significantly in terms of query execution times.

## References

1. Tao, Y., Papadias, D., Lian, X.: Reverse kNN search in arbitrary dimensionality. In: Proc. VLDB (2004)
2. Lazaridis, I., Mehrotra, S.: Progressive approximate aggregate queries with a multi-resolution tree structure. In: Proc. SIGMOD (2001)
3. Papadias, D., Kalnis, P., Zhang, J., Tao, Y.: Efficient olap operations in spatial data warehouses. In: Proc. SSTD (2001)
4. Guttman, A.: R-Trees: A dynamic index structure for spatial searching. In: Proc. SIGMOD, pp. 47–57 (1984)
5. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-Tree: An efficient and robust access method for points and rectangles. In: Proc. SIGMOD, pp. 322–331 (1990)
6. Achtert, E., Bähm, C., Kröger, P., Kunath, P., Pryakhin, A., Renz, M.: Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In: Proc. SIGMOD (2006)
7. Achtert, E., Böhm, C., Kröger, P., Kunath, P., Pryakhin, A., Renz, M.: Approximate reverse k-nearest neighbor search in general metric spaces. In: Proc. CIKM (2006)
8. Achtert, E., Böhm, C., Kröger, P., Kunath, P., Pryakhin, A., Renz, M.: Efficient reverse k-nearest neighbor estimation. In: Proc. BTW (2007)
9. Tao, Y., Yiu, M.L., Mamoulis, N.: Reverse nearest neighbor search in metric spaces. IEEE TKDE 18(9), 1239–1252 (2006)
10. Korn, F., Muthukrishnan, S.: Influenced sets based on reverse nearest neighbor queries. In: Proc. SIGMOD (2000)
11. Yang, C., Lin, K.I.: An index structure for efficient reverse nearest neighbor queries. In: Proc. ICDE (2001)
12. Singh, A., Ferhatosmanoglu, H., Tosun, A.S.: High dimensional reverse nearest neighbor queries. In: Proc. CIKM (2003)
13. Stanoi, I., Agrawal, D., Abbadi, A.E.: Reverse nearest neighbor queries for dynamic databases. In: Proc. DMKD (2000)
14. Berchtold, S., Keim, D.A., Kriegel, H.P.: The X-Tree: An index structure for high-dimensional data. In: Proc. VLDB (1996)

# Finding Structural Similarity in Time Series Data Using Bag-of-Patterns Representation

Jessica Lin and Yuan Li

Computer Science Department
George Mason University
`jessica@cs.gmu.edu, ylif@gmu.edu`

**Abstract.** For more than one decade, time series similarity search has been given a great deal of attention by data mining researchers. As a result, many time series representations and distance measures have been proposed. However, most existing work on time series similarity search focuses on finding shape-based similarity. While some of the existing approaches work well for short time series data, they typically fail to produce satisfactory results when the sequence is long. For long sequences, it is more appropriate to consider the similarity based on the higher-level structures. In this work, we present a histogram-based representation for time series data, similar to the "bag of words" approach that is widely accepted by the text mining and information retrieval communities. We show that our approach outperforms the existing methods in clustering, classification, and anomaly detection on several real datasets.

**Keywords:** Data mining, Time series, Similarity Search.

## 1 Introduction

Time series similarity search has been a major research topic for time series data mining for more than one decade. As a result, many time series representations and distance measures have been proposed [3, 6, 12, 15, 17, 19]. There are two kinds of similarities: shape-based similarity and structure-based similarity [13]. The former determines the similarity of two datasets by comparing their local patterns, whereas the latter determines the similarity by comparing their global structures. Most existing approaches focus on finding shape-based similarity. While some of these approaches work well for short time series data, they typically fail to produce satisfactory results when the sequence is long. To understand the need for a higher-level, structure-based similarity measure for long time series data, consider the scenario for textual data. If we are to compare two strings, we can use the string edit distance to compute their similarity. However, if we want to compare two documents, we typically do not compare them on the word-to-word basis. Instead, it is more meaningful to use a higher-level representation that can capture the structure or semantic of the document. Below, we describe the two types of similarities in more detail [13].

Given two sequences A and B, shape-based similarity determines how similar these two datasets are based on local comparisons. The most well-known distance measure in data mining literature is the Euclidean distance, for which sequences are aligned in the point-to-point fashion, i.e. the ith point in sequence A is matched with the ith point in sequence B. While Euclidean distance works well in general, it does not always produce accurate results when data is shifted, even slightly, along the time axis. For example, the top and bottom sequences in Figure 1 appear to have similar shapes. In fact, the sequence below is the shifted version of the sequence above. However, the slight shifts along the time axis will result in a large distance between the two sequences.

Another distance measure, Dynamic Time Warping (DTW) [12, 24], overcomes this limitation by using dynamic programming technique to determine the best alignment that will produce the optimal distance. The parameter, warping length, determines how much warping is allowed to find the best alignment. A large warping window causes the search to become prohibitively expensive, as well as possibly allowing meaningless matching between points that are far apart. On the other hand, a small window might prevent us from finding the best solution. Euclidean distance can be seen as a special case of DTW, where there is no warping allowed. Figure 1 demonstrates the difference between the two distance measures. Note that with Euclidean distance, the dips and peaks in the sequences are mis-aligned and therefore not matched, whereas with DTW, the dips and peaks are aligned with their corresponding points from the other sequence. While DTW is a more robust distance measure than Euclidean distance, it is also a lot more computationally intensive. Keogh [12] proposed an indexing scheme for DTW that allows faster retrieval. Nevertheless, DTW is still at least several orders slower than Euclidean distance.

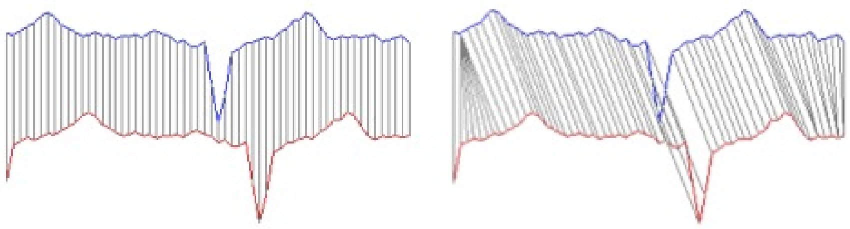


**Fig. 1.** (Left) Alignment for Euclidean distance between two sequence data. (Right) Alignment for Dynamic Time Warping distance between two sequence data.

Shape-based similarities work well for short time series or subsequences; however, for long sequence data, they produce poor results. To illustrate this, we extracted subsequences of length 682 from six different records on PhysioNet [10], an online medical archive containing digital recordings of physiological signals. The first three datasets (numbered #1 ~ #3) are measurements on respiratory rates, and the last three datasets (#4 ~ #6) are electrocardiograms (ECGs). As we can clearly see in Figure 2, these two types of vital signs have very different structures. Visually we can separate the two classes with no difficulty. However, if we cluster them using Euclidean distance as the distance measure, the result is disappointing. Figure 2 shows the hierarchical clustering result using Euclidean distance. The dendrogram illustrates that while datasets #5 and #6 are correctly clustered (i.e. they share a common parent node), the rest are not.

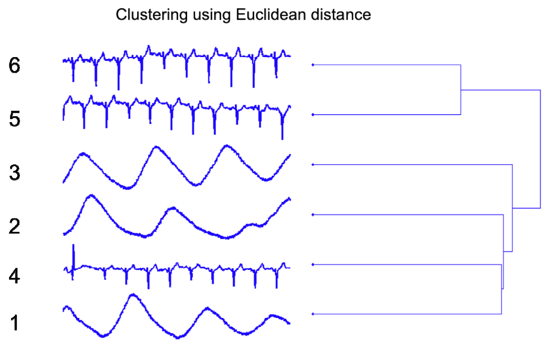


**Fig. 2.** Result of hierarchical clustering using Euclidean distance on raw data. In fact, using DTW produces the same result.

One reason for the poor clustering result could be that the datasets within the same cluster are not perfectly aligned. In addition, the presence of anomalous points, as shown in the beginning of dataset #4, could also throw off the distances computed. DTW can be used to mitigate the first issue to a certain extent. However, in this example, DTW does not seem to offer any improvement; clustering using DTW produces a similar dendrogram as the one shown in Figure 2. Furthermore, the high computational cost for dynamic time warping makes it a less desirable choice of distance measure for large datasets.

A more appropriate alternative to determine similarity between long sequences is to measure their similarity based on higher-level structures. Several structure- or model-based similarities have been proposed that extract global features such as trend, autocorrelation, skewness, and model parameters from data [23, 26]. However, it is not trivial how to determine relevant features, and/or compute distances given these features [13]. In addition, often these global features are not sufficient enough to capture the information needed in order to distinguish between the data.

In this paper, we focus on finding structural similarities between time series data. Our method is robust and efficient, and it is inspired by the well-known bag-of-words representation for text data. There are several advantages for our approach compared to existing structure-based method. First, since the overall representation is built from extracting the subsequences from data, we in fact take local structures into consideration as well as global structures. Furthermore, the incremental construction of the representation suggests that it can be used in the streaming data scenario. Our representation also allows users to understand the pattern distribution of the data by examining the resulting histograms. We show that our approach outperforms existing methods in the tasks of classification, clustering, and anomaly detection on real datasets.

The rest of the paper is organized as follows. In Section 2 we briefly discuss background and related work. Section 3 presents our methodology. In Section 4, we show empirical results in clustering, classification, and anomaly detection. We conclude and discuss future work in Section 5.

## 2 Background and Related Work

In this section, we briefly discuss background and related work on time series similarity search. For concreteness, we begin with a definition of time series:

**Definition 1**. *Time Series*: A time series $T = t_1,\ldots,t_p$ is an ordered set of $p$ real-valued variables.

Some distance measure *Dist*(*A*,*B*) needs to be defined in order to determine the similarity between time series objects A and B.

**Definition 2**. *Distance*: *Dist* is a function that has *A* and *B* as inputs and returns a nonnegative value *R*, which is said to be the distance from *A* to *B*.

Each time series is normalized to have a mean of zero and a standard deviation of one before calling the distance function, since it is well understood that in virtually all settings, it is meaningless to compare time series with different offsets and amplitudes [15].

As mentioned, Euclidean distance and Dynamic Time Warping are among the most commonly used distance measures for time series. For this reason, we will use Euclidean distance for our new representation, and compare the results with using Euclidean distance and Dynamic Time Warping on raw data.

In addition to comparing with well-known distance measures on the *raw* data, we also demonstrate that our method outperforms existing time series representations such as Discrete Fourier Transform (DFT) [1]. DFT approximates the signal with a linear combination of basis functions, and its coefficients represent global contribution of the signal. Another well-known representation is Discrete Wavelet Transform (DWT) [3]. Wavelets are mathematical functions that represent data or other functions in terms of the averages and differences of a prototype function, called the analyzing or mother wavelet. Contrary to DFT, wavelets are localized in time. Nevertheless, past studies have shown that DFT and DWT have similar performance in terms of accuracy [15].

While there have been dozens of representations and distance measures proposed for time series shape-based similarity, there is relatively little work on finding structure-based similarity. Deng et al [4] proposed learning ARMA model on the time series, and using the model coefficients as the feature. This approach has an obvious limitation on the characteristics of input data. Ge and Smyth [8] proposed a deformable Markov Model template for temporal pattern matching, in which the data is converted to a piecewise linear model. However, this approach requires many parameters. Nanopoulos et al [23] proposed extracting statistical features of time series such as skewness, mean, variance, and kurtosis, and classifying the data using multi-layer perceptron (MLP) neural network.

Recently, Keogh et al. [16] proposed a compression-based distance measure that compares the co-compressibility between datasets. Motivated by Kolmogorov Complexity [16, 19] and promising results shown in similar work in bioinformatics and computational theory, the authors devised a new dissimilarity measure called CDM (Compression-based Dissimilarity Measure). Given two datasets (strings) *x* and *y*, their compression-based dissimilarity measure can be formulated as follows:

$$CDM(x,y) = \frac{C(xy)}{C(x)+C(y)}$$

where *C(xy)* is the compressed size of the concatenated string *x+y*, *C(x)* and *C(y)* are the compressed sizes of the string *x* and *y*, respectively. In their paper, the authors show superior results compared to other existing structural similarity approaches. For this reason, we will compare our method with CDM, the best structure-based (dis)similarity measure reported. We will show that our approach is highly competitive, with several additional advantages over existing methods.

## 3 Finding Structural Similarity

We propose a histogram-based similarity measure, using a representation similar to the one widely used for text data. In the Vector Space Model [25], each document can be represented as a vector in the vector space. Each dimension of the vector corresponds to one word in the vocabulary, and the value of each component is the (relative) frequency of occurrences for the corresponding word in the document. As a result, an *p*-by-*q* term-to-document matrix *X* is constructed, where *p* is the number of unique terms in the text collection, *q* is the number of documents, and each element *X(i,j)* is the frequency of the $i^{th}$ word occurring in the $j^{th}$ document.

This "bag of words" representation works well for documents. It is able to capture the structure or topic of a document, without knowing the exact locations or orderings of the word appearances. This suggests that we might be able to represent time series data in a similar fashion, i.e. as a combination of patterns from a finite set of patterns.

There are two challenges if we wish to represent time series data as a "bag of patterns." The first challenge concerns with the definition and construction of the patterns "vocabulary." The second challenge comes from the fact that time series data are composed of consecutive data points. There is no clear "delimiters" between patterns. Fortunately, a symbolic representation for time series called SAX (Symbolic Aggregate approXimation) [20], previously developed by the current author and now a widely used discretization method, provides solutions to these challenges. The intuition is to convert the time series into a set of SAX words, and then construct a word-sequence matrix (analogous to the term-document matrix for text data) using these SAX words. In the next section, we briefly discuss how SAX converts time series data into strings.

### 3.1 Symbolic Aggregate approXimation

Given a time series of length *n*, SAX produces a lower dimensional representation of a time series by transforming the original data into symbolic words. Two parameters are used to specify the size of the alphabet to use (i.e. $\alpha$) and the size of the words to produce (i.e. *w*). The algorithm begins by using a normalized version of the data and creating a Piecewise Aggregate Approximation (PAA). PAA reduces the dimensionality of a time series by transforming the original representation into a user defined number (i.e. *w*, typically $w << n$) of equal segments. The segment values are determined by calculating the mean of the data points in that segment. The PAA values are then transformed into symbols by using a breakpoint table. The breakpoints in the breakpoint table are defined such that all the regions have approximately equi-probability based on a Gaussian distribution[1]. These breakpoints may be determined by looking them up in a statistical table. For example, Table 1 gives the breakpoints for values of α from 3 to 5.

**Table 1.** A lookup table that contains the breakpoints that divides a Gaussian distribution into an arbitrary number (from 3 to 5) of equiprobable regions

| $\beta_i$ \ $a$ | 3 | 4 | 5 |
|---|---|---|---|
| $\beta_1$ | -0.43 | -0.67 | -0.84 |
| $\beta_2$ | 0.43 | 0 | -0.25 |
| $\beta_3$ | | 0.67 | 0.25 |
| $\beta_4$ | | | 0.84 |

Figure 3 summarizes how a time series is converted to PAA and then symbols, with parameters α = 3 and $w$ = 8.

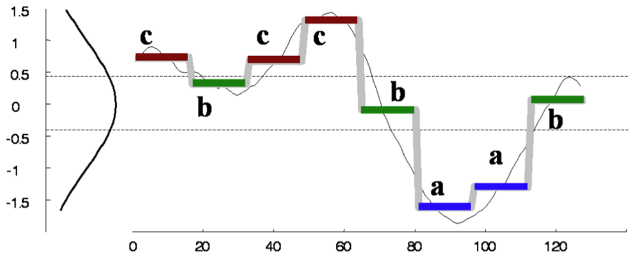


**Fig. 3.** Example of SAX for a time series, with parameters α = 3 and $w$ = 8. The time series above is transformed to the string *cbccbaab*, and the dimensionality is reduced from 128 to 8.

## 3.2 Bag-of-Words Representation for Time Series

Our algorithm works as follows. For each time series, we use a sliding window and extract every possible subsequence of length $n$ (a user-defined parameter). Each subsequence is normalized to have mean of zero and standard deviation of one before it is converted to a SAX string. As a result, we obtain a set of strings, each of which corresponds to a subsequence in the time series. As noted in [20], given a subsequence $S_i$, it is likely to be very similar to its neighboring subsequences, $S_{i-1}$ and $S_{i+1}$ (i.e. those that start one point to the left, and one point to the right of $S_i$), especially if $S_i$ is in the smooth region of the time series. These subsequences are called *trivial matches* of $S_i$. To avoid over-counting these trivial matches as true

[1] The Gaussian assumption is the default, since in [20] the authors discover that most short time series subsequences follow the Gaussian distribution. However, the breakpoints can be adjusted based on the actual distribution of the data.

patterns, we need to perform numerosity reduction. Since SAX preserves the general shape of the sequence, in some cases we might see that multiple consecutive subsequences are mapped to the same string. In that case, we record only the first occurrence of the string, and ignore the rest until we encounter a string that is different. In other words, for each group of consecutive identical strings, we record only the first occurrence and count this group of occurrences only once.

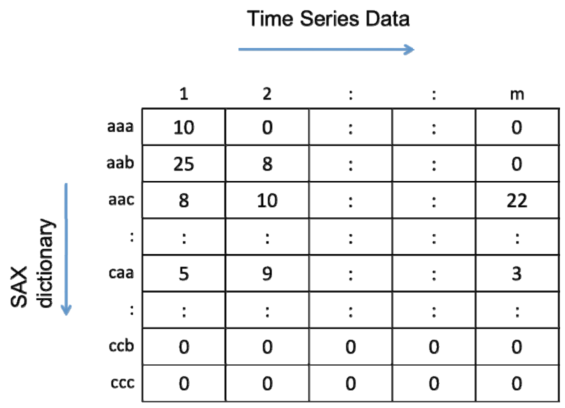
Once we obtain the set of strings for each time series dataset, we can construct the word-sequence matrix. Given $\alpha$ and $w$ as the parameters for SAX, we know the size of our entire collection of possible SAX strings, or our "dictionary." There are $\alpha^w$ possible SAX words. For example, for $\alpha = 4$ and $w = 4$, our dictionary size is only 256. Clearly, the size of the dictionary increases exponentially with the increase of $w$. Experimental results in previous work [20] indicate that the choice of $\alpha$ does not critically affect the performance. Typically, a value of 3 or 4 works well for most time series datasets. In this paper, we choose $\alpha = 4$.

Having fixed $\alpha$, we now have to determine the value for $w$. While the best choice of $w$ is data-dependent, generally speaking, time series with smooth patterns can be described with a small $w$, and those with rapidly changing patterns prefer large $w$ to capture the critical changes. We choose $w$ = 6~8 for our experiments, with sliding window length of 100~300.

With $\alpha = 4$ and $w = 8$, the resulting dictionary size is $\alpha^w = 4^8 = 65536$, which is still quite large. However, as is the case for text documents, the matrix is likely to be sparse. Therefore, we can eliminate the words that never occur in the data, and/or store the matrix in a compressed format such as the Compressed Column Storage (CCS) format [5]. In our experiments, we find that only about 10% of all words have some subsequence mapped to it.

The construction of the temporal "bag of patterns" matrix is straightforward. The matrix M is a word-sequence matrix, where each row $i$ denotes a SAX word (i.e. a pattern) from the dictionary; each column $j$ denotes a time series dataset; and each $M_{i,j}$ stores the frequency of word $i$ occurring in time series $j$. Within each matrix cell $M_{i,j}$, we can also store a list of pointers to the subsequence in time series $j$ (typically on disk) that are mapped to word $i$. The lists of pointers will enable us to perform subsequence matching. However, since we are focused on finding structural similarities between time series data, we will contend ourselves with storing just the word frequency in the matrix for now. We call this new representation BOP (Bag of Patterns).

The matrix provides a summary of time series data in terms of the frequency of occurrence for each pattern. Once we build the matrix M, we can then use any applicable distance measures or dimensionality reduction techniques to computes the similarity between different time series datasets. Figure 4 shows a visual example of this representation. Like the bag-of-words representation for documents, the orderings of words are lost. However, for long time series data, this level of details is exactly the reason why conventional shaped-based approaches do not work well. As our experiments demonstrate, BOP produces very good results even without knowing the ordering of the patterns.


Time Series Data
SAX dictionary

| | 1 | 2 | : | : | m |
|---|---|---|---|---|---|
| aaa | 10 | 0 | : | : | 0 |
| aab | 25 | 8 | : | : | 0 |
| aac | 8 | 10 | : | : | 22 |
| : | : | : | : | : | : |
| caa | 5 | 9 | : | : | 3 |
| : | : | : | : | : | : |
| ccb | 0 | 0 | 0 | 0 | 0 |
| ccc | 0 | 0 | 0 | 0 | 0 |

**Fig. 4.** A visual example of the bag-of-patterns representation for time series. Each row denotes a SAX word, and each column denotes a time series data. We could also store, within each cell, pointers to corresponding subsequences.

## 4 Empirical Evaluation

In this section, we present empirical evaluation of our method on clustering, classification, and anomaly detection.

### 4.1 Clustering

For this part of experiments, we demonstrate the effectiveness of our approach in clustering. We show that our representation out-perform existing approaches and produce more accurate clustering results.

#### *4.1.1 Hierarchical Clustering*

One of the most widely used clustering approaches is hierarchical clustering [11]. Hierarchical clustering computes pairwise distances of the objects (or groups of objects) and produces a nested hierarchy of the clusters. It has several advantages over other clustering methods. More specifically, it offers great visualization power with the hierarchy of clusters, and it requires no input parameters. However, its intensive computational complexity makes it infeasible for large datasets.

In Figure 2, we showed a simple example on hierarchical clustering where both Euclidean Distance and Dynamic Time Warping on the raw data fail to find the correct clusters. In this part of experiment, as a sanity check, we show the clustering result using BOP to represent the time series datasets. We use Euclidean distance to compute the similarity between the histograms (i.e. the column vectors). Figure 5 shows the resulting dendrogram. Note we are now clustering on the transformed time series, or the histogram of the patterns. For clarity, we also plot the original, corresponding time series to the left of the histograms. We can see

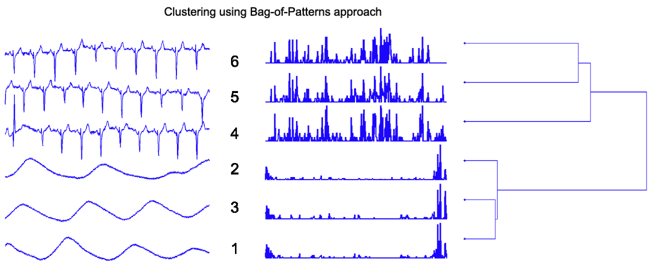


**Fig. 5.** New clustering result on the same data shown in Figure 2. This time, we use our histogram-based, bag-of-patterns approach, and combine it with Euclidean distance. The two clusters are well separated.

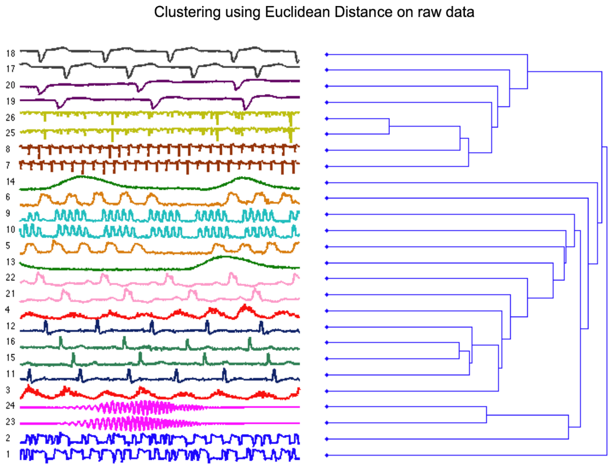


**Fig. 6.** Clustering result using Euclidean distance on the raw data. Only three pairs of data are cleanly clustered together.

clearly from the histograms that the time series clustered together share similar pattern distribution.

While the example shown above gives us a first indication that our approach can find clusters while shape-based approaches cannot, with only six datasets, the example is too small and contrived to offer any conclusive insight. Therefore, we perform more hierarchical clustering experiments on larger datasets. We compare our approach with the following methods: (1) CDM proposed by Keogh et al [16], (2) Euclidean distance on raw time series, (2) Dynamic Time Warping on raw time series, and (4) Euclidean distance on DFT coefficients.

Similar to the experimental setting in [16], we selected 13 pairs of datasets across different domains, with diverse structures from UCR Time Series Archive [14]. Although our method does not require the input time series to have the same length, for simplicity, we keep each dataset at length 1000. For Dynamic Time Warping, however, since it is costly to compute, we reduce the time series length to 250 by systematic sampling.

Figure 6[2] shows the dendrogram produced by hierarchical clustering using Euclidean distance on the raw data. We can see that out of 13 pairs of datasets, only 3 pairs are successfully clustered together.

As illustrated in Figure 7, DTW shows improvement over Euclidean distance and successfully clusters 10 pairs of dataset. Perhaps the clustering results for DTW would have been better if we used the entire dataset. However, its prohibitive time complexity makes it an unrealistic choice of distance measure for large datasets. Even at the reduced length, DTW took 156 times longer than our approach on the *whole* dataset.

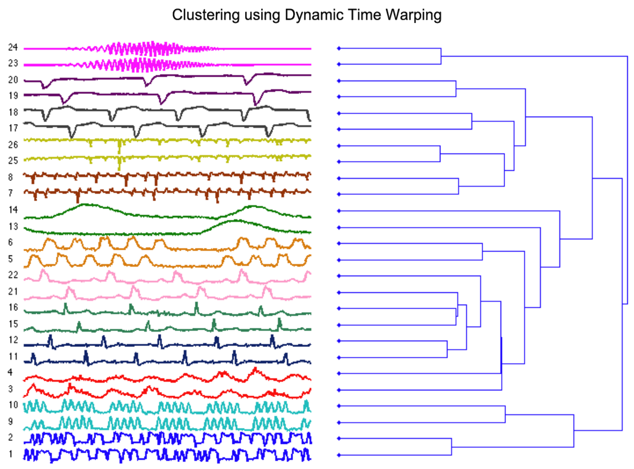


**Fig. 7.** Clustering result using Dynamic Time Warping on the raw data. The result is better than Euclidean: 10 pairs of data are cleanly clustered together.

Next, we convert the time series by DFT, and cluster the data on the DFT coefficients. One of the advantages of DFT is that it offers dimensionality reduction. As demonstrated in [1], most "energy" concentrates on the first few DFT coefficients. Therefore, we can use only a few DFT coefficients to approximate the data, while still preserving the general shape of the data. If we use all the coefficients, then we get back the original sequence. In this experiment, we used 100 coefficients (compared to 1000 data points in the raw data). Similar to using the raw data, only 3 pairs of data are cleanly clustered. Figure 8 shows the result.

Figure 9 shows the clustering result produced by our BOP approach. Again, the histograms for the bag-of-patterns are shown in the middle of the figure. As we can see, all 13 pairs are successfully clustered.

[2] The figures are best viewed in color.

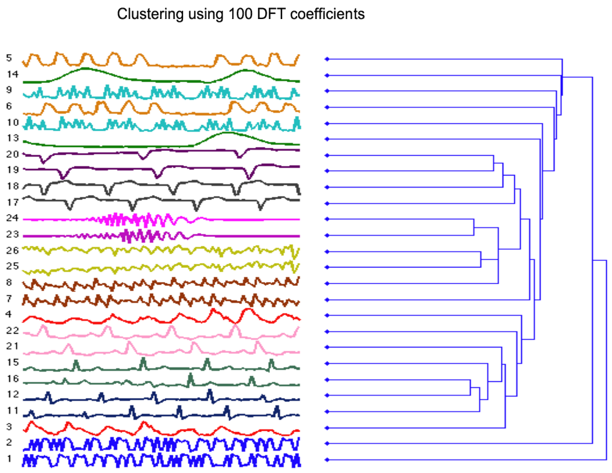


**Fig. 8.** Clustering results using 100 DFT coefficients. Only three pairs are correctly clustered together.

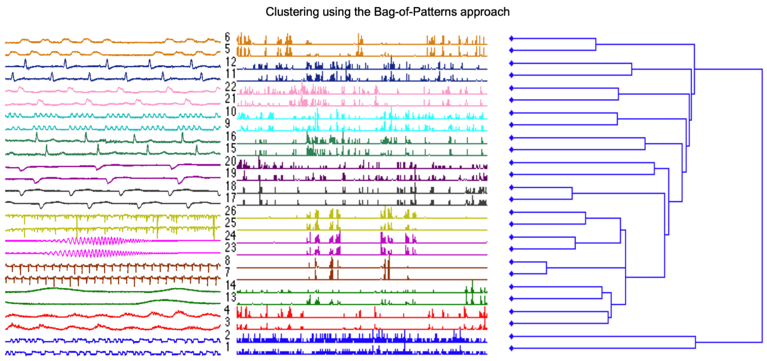


**Fig. 9.** Clustering results using our approach. All pairs of data are successfully clustered.

The results above are promising. However, these time series are still relatively short, and they have very diverse structures. To see how our algorithm does on long sequences, and on data with less diverse structures, we performed hierarchical clustering on the ECG dataset presented in [16]. We will call this dataset ECG1. This dataset contains 20 ECG records that form 4 clusters. Details on the datasets can be found in [16]. Each record is of length 15,000. Our results are comparable to that reported in [16]. Regardless of the high level of noises in the data, all 4 clusters are correctly identified, as demonstrated in Figure 10.

While CDM produces similar results, our approach offers several advantages. First, since we cluster on the pattern histograms, we can see the distribution of patterns from these histograms, and understand the underlying structures of the data. Furthermore, since we extract subsequences and use them to build the final representation, our approach is potentially suitable for streaming data.

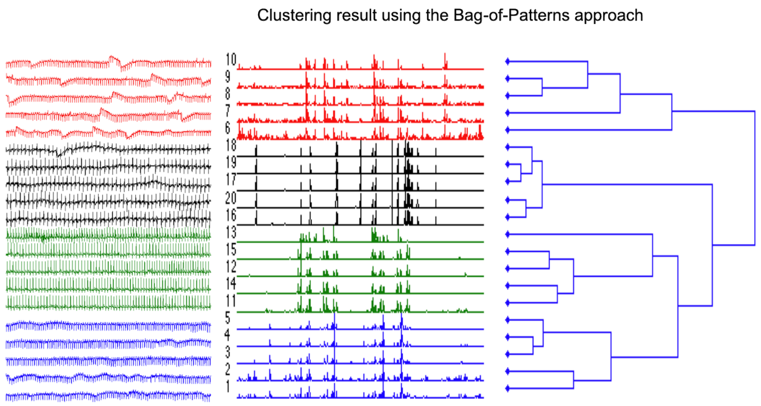


**Fig. 10.** Clustering result on 20 ECG datasets, using our bag-of-patterns approach. Each record is 15,000 points long.

Next, we compare our results with the three other methods that we mentioned. Figure 11 shows the clustering results using Euclidean distance on the raw data. Only 9 out of 20 datasets are clustered correctly.

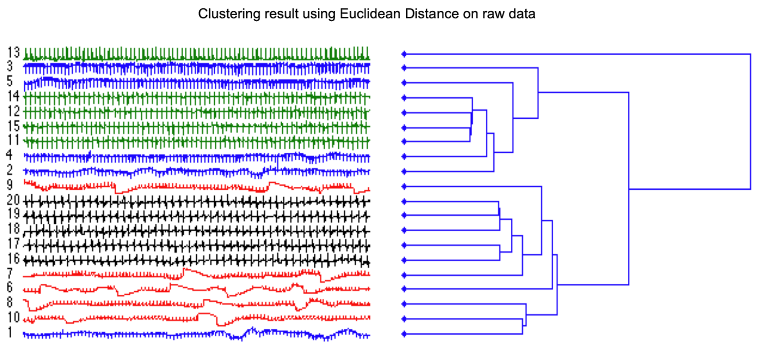


**Fig. 11.** Clustering result on raw ECG1 data using Euclidean Distance. Only 9 datasets are correctly clustered (#11, #12, #14, #15, #16-#20).

When we repeat the experiment using DTW, we had to sample down the data (20:1), due to its high computational cost. Our machine simply could not handle it. With DTW on the shorter datasets, the result is similar to that of Euclidean distance on the full datasets. To show that this poor result is not just due to the loss of data from sampling, we re-ran the experiment using our BOP representation on the sampled, shorter datasets, and obtained the same result as shown in Figure 10.

For the final comparison, we convert the time series to DFT coefficients, and cluster the data on the coefficients. Similar to Euclidean Distance on the raw data, only 8 datasets are clustered correctly. We also tried different resolutions (100-1000 coefficients), but obtained poor results regardless of the resolution. The result is shown in Figure 12.

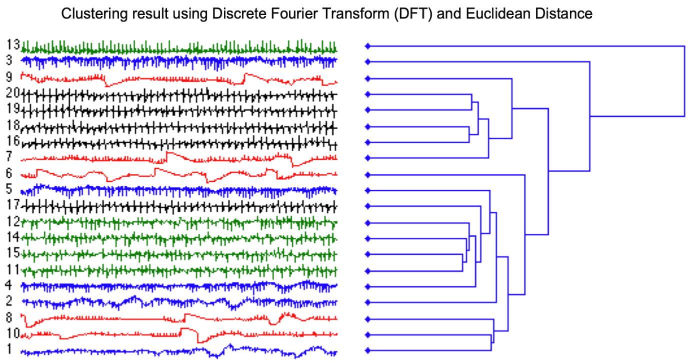


**Fig. 12.** Clustering result on ECG1 data using 1000 DFT coefficients

### *4.1.2 Partitional Clustering*

Although hierarchical clustering is a good sanity check from its visualization power, it has limited utility due to its poor scalability. The most commonly used data mining clustering algorithm is *k*-means [2, 22, 21]. We performed *k*-means using the Euclidean distance on the raw data, and on our bag-of-patterns representation. The basic intuition behind *k*-means (and in general, iterative refinement algorithms) is the continuous reassignment of objects into different clusters, so that the intra-cluster distance is minimized.

We performed *k*-means using the Euclidean distance on the raw data, and on our histogram-based representation. CDM is not included in this experiment, as it's unclear how to define the centroid of a cluster [16].

For this experiment, we extracted 250 records from the PhysioNet archive. Each record contains 2048 points. These records are extracted from various databases containing different vital signs, or patients with different heart conditions. We separated the records into 5 classes, and labeled them according to the databases that they are extracted from. We will call this dataset ECG2. Figure 13 shows one example from each of the 5 classes in ECG2 dataset.

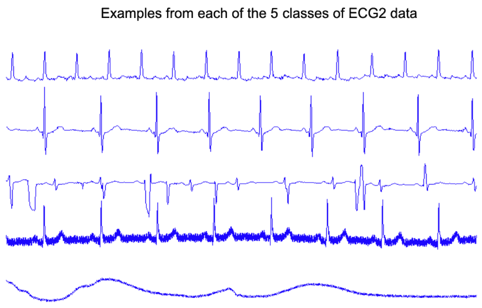


**Fig. 13.** One example from each of the 5 classes in ECG2 dataset

We ran *k*-means algorithm 10 times, and recorded the clustering labels obtained from the run with the smallest objective function (i.e. sum of intra-cluster distances). We then compare our cluster labels with the true labels, and compute the clustering quality using the evaluation method proposed by [7]. The evaluation method compares the similarity between two sets of cluster labels, and returns a number between 0 and 1 denoting how similar they are. Ideally, we would like the number to be as close to 1 as possible. Our approach achieves the best clustering quality (0.7133 vs. 0.4644). The results are shown in Table 2.

## 4.2 Classification

Classification of time series has attracted much interest from the data mining community [10, 22, 23, 24]. For the classification experiments, we will consider the most common classification algorithm, nearest neighbor classification. To demonstrate the effectiveness on 1-nearest-neighbor classification, we use the same ECG2 dataset. We use the leave-one-out cross validation, and count the number of correctly classified objects, *cc*. The accuracy is the ratio of *cc* and the total number of objects (i.e. 250). For this experiment, we also add Dynamic Time Warping (again, with reduced length). The accuracy results are show in Table 2. The improvement is astounding. For our approach, the accuracy of 0.996 means that there is only 1 misclassified object, out of 250 objects.

## 4.3 Discord/Anomaly Detection

In [18], a discord is defined as the data object that is the least similar to the rest of the dataset, i.e. it has the largest nearest neighbor distance. A discord can be seen as an anomaly in the data. In this section, we conduct discord/anomaly detection experiments, using our BOP representation and comparing it with Euclidean distance on the raw data. We use the same ECG2 dataset. We take one class of ECG2 data at a time (each class contains 50 ECG records), and manually insert an anomaly by randomly choosing one other ECG record that belongs to a different class. For each class, we repeat this experiment 20 times, which results in a total of 100 runs. We then compare the accuracy, or the percentage of discords found, of our representation against the raw data, both using Euclidean distance. The accuracy results are shown in Table 2.

**Table 2.** Accuracy of our approach on clustering, classification, and discord discovery compared to other methods. Our approach achieves the best accuracy for all tasks. All numbers are between 0 and 1.

| | Euclidean | DTW | Bag-of-Patterns |
|---|---|---|---|
| k-means | 0.4644 | N/A | **0.7133** |
| NN | 0.44 | 0.728 | **0.996** |
| Discord | 0.35 | N/A | **0.85** |

One of the reasons that our approach works so much better than using the raw data is that the ECG data are not at all aligned, even for datasets in the same class. Another reason is that sometimes an ECG data might contain local anomalies within the data; such anomalies can easily throw off the distances computed using the shape-based approach.

We conclude this experiment by noting that the definition of discord can be easily extended to top *k*-discords. Applying *k*-discords discovery algorithm will allow us to find both global (i.e. data that do not belong in the cluster) and local (i.e. anomalies that occur within one time series) anomalies.

## 5 Conclusion

Most existing work on time series similarity search focuses on finding shaped-based similarity. While these shape-based approaches work reasonably well for short time series data, the accuracy typically degrades if the sequences are long. For long time sequences, it is more appropriate to measure the similarity by looking at their higher-level structures, rather than point-to-point, local comparisons.

In this work, we proposed a histogram-based similarity measure, BOP. Similar to the bag-of-words representation for textual data, our approach counts the frequency of occurrences of each pattern in the time series. We then compare the frequencies (or the histograms) of patters in the time series to obtain a (dis)similarity measure.

Our experimental results show that our approach is superior to existing approaches in the tasks of clustering, classification, and anomaly detection. Furthermore, our approach has several advantages over existing structure-based similarity measures. Specifically, our approach considers local structures as well as global structure, by using subsequences to build our final representation. Our representation allows users to understand the pattern distribution by examining the histograms. Furthermore, our representation is suitable for streaming data, since the frequency vectors are built incrementally.

We would like to note that since our approach determines similarity based on structures of the data, the input sequences should be reasonably long, or long enough such that the structures (or lack of structures) can be meaningfully captured and summarized.

For future work, we will explore our representation on other types of data such as images. We will also try other existing distance measures such as cosine similarity, and/or devise a distance measure more suitable than Euclidean Distance.

**Acknowledgments.** We would like to thank Eamonn Keogh for providing the datasets and the code for CDM.

## References

1. Agrawal, R., Faloutsos, C., Swami, A.: Efficient Similarity Search in Sequence Databases. In: Proceedings of the 4th Int'l Conference on Foundations of Data Organization and Algorithms, Chicago, IL, October 13-15, pp. 69–84 (1993)
2. Bradley, P., Fayyad, U., Reina, C.: Scaling Clustering Algorithms to Large Databases. In: Proceedings of the 4th Int'l Conference on Knowledge Discovery and Data Mining, New York, NY, August 27-31, pp. 9–15 (1998)

3. Chan, K., Fu, A.W.: Efficient Time Series Matching by Wavelets. In: Proceedings of the 15th IEEE Int'l Conference on Data Engineering, Sydney, Australia, March 23-26, pp. 126–133 (1999)
4. Deng, K., Moore, A., Nechyba, M.: Learning to Recognize Time Series: Combining ARMA models with Memory-based Learning. In: IEEE Int. Symp. on Computational Intelligence in Robotics and Automation, vol. 1, pp. 246–250 (1997)
5. Ekambaram, A., Montagne, E.: An Alternative Compressed Storage Format for Sparse Matrices. In: Yazıcı, A., Şener, C. (eds.) ISCIS 2003. LNCS, vol. 2869, pp. 196–203. Springer, Heidelberg (2003)
6. Faloutsos, C., Ranganathan, M., Manolopulos, Y.: Fast Subsequence Matching in Time-Series Databases. SIGMOD Record 23, 419–429 (1994)
7. Gavrilov, M., Anguelov, D., Indyk, P., Motwahl, R.: Mining the stock market: which measure is best? In: Proc. of the 6th ACM SIGKDD (2000)
8. Ge, X., Smyth, P.: Deformable Markov model templates for time-series pattern matching. In: Proceedings of the 6th ACM SIGKDD, Boston, MA, August 20-23, pp. 81–90 (2000)
9. Geurts, P.: Pattern Extraction for Time Series Classification. In: Siebes, A., De Raedt, L. (eds.) PKDD 2001. LNCS, vol. 2168, pp. 115–127. Springer, Heidelberg (2001)
10. Goldberger, A.L., Amaral, L., Glass, L., Hausdorff, J.M., Ivanov, P.C., Mark, R.G., Mietus, J.E., Moody, G.B., Peng, C.K., Stanley, H.E.: PhysioBank, PhysioToolkit, and PhysioNet: Circulation. Discovery 101(23), 1(3), e215–e220 (1997)
11. Johnson, S.C.: Hierarchical Clustering Schemes. Psychometrika 2, 241–254 (1967)
12. Keogh, E.: Exact indexing of dynamic time warping. In: Proceedings of the 28th international Conference on Very Large Data Bases, Hong Kong, China, August 20-23 (2002)
13. Keogh, E.: Tutorial in SIGKDD 2004. Data Mining and Machine Learning in Time Series Databases (2004)
14. Keogh, E., Folias, T.: The UCR Time Series Data Mining Archive. Riverside CA (2002), `http://www.cs.ucr.edu/~eamonn/TSDMA/index.html`
15. Keogh, E., Kasetty, S.: On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery, Edmonton, Alberta, Canada, pp. 102–111 (2002)
16. Keogh, E., Lonardi, S., Ratanamahatana, C.A.: Towards parameter-free data mining. In: Proceedings of the Tenth ACM SIGKDD international Conference on Knowledge Discovery and Data Mining, KDD 2004, Seattle, WA, USA, August 22 - 25 (2004)
17. Keogh, E., Chakrabarti, K., Pazzani, M.: Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In: Proceedings of ACM SIGMOD Conference on Management of Data, Santa Barbara, May 21-24, pp. 151–162 (2001)
18. Keogh, E., Lin, J., Fu, A.: Finding the Most Unusual Time Series Subsequence: Algorithms and Applications. In: Knowledge and Information Systems (KAIS). Springer, Heidelberg (2006)
19. Li, M., Vitanyi, P.: An Introduction to Kolmogorov Complexity and Its Applications, 2nd edn. Springer, Heidelberg (1997)
20. Lin, J., Keogh, E., Li, W., Lonardi, S.: Experiencing SAX: A Novel Symbolic Representation of Time Series. Data Mining and Knowledge Discovery Journal (2007)
21. Lin, J., Vlachos, M., Keogh, E., Gunopulos, D.: Iterative Incremental Clustering of Time Series. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 106–122. Springer, Heidelberg (2004)

22. McQueen, J.: Some Methods for Classification and Analysis of Multivariate Observation. In: Le Cam, L., Neyman, J. (eds.) Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, CA, vol. 1, pp. 281–297 (1967)
23. Nanopoulos, A., Alcock, R., Manolopoulos, Y.: Feature-based classification of time-series data. In: Mastorakis, N., Nikolopoulos, S.D. (eds.) Information Processing and Technology, pp. 49–61. Nova Science Publishers, Commack (2001)
24. Ratanamahatana, C.A., Keogh, E.: Making Time-series Classification More Accurate Using Learned Constraints. In: Proceedings of SIAM International Conference on Data Mining (SDM 2004), Lake Buena Vista, Florida, April 22-24 (2004)
25. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Commun. ACM 19(11), 613–620 (1975)
26. Wang, X., Smith, K., Hyndman, R.: Characteristic-Based Clustering for Time Series Data. Data Min. Knowl. Discov. 13(3), 335–364 (2006)

# Cloud Computing for Science

Kate Keahey

Argonne National Laboratory
keahey@mcs.anl.gov

Infrastructure-as-a-Service (IaaS) style cloud computing is emerging as a viable alternative to the acquisition and management of physical resources. This raises several questions. How can we take advantage of the opportunities it offers? Are the current commercial offerings suitable for science? What cloud capabilities are required by scientific applications? What additional infrastructure is needed to take advantage of cloud resources?

In this talk, I will describe several application projects using cloud computing in commercial and academic space and discuss the challenges and benefits of this approach in terms of performance, cost, and ease-of-use. I will also discuss our experiences with configuring and running the Science Clouds – a group of clouds in academic domain available to scientific projects configured using the Nimbus Toolkit. Finally, I will discuss the emerging trends and innovation opportunities in cloud computing.

# Classification with Unknown Classes

Chetan Gupta, Song Wang, Umeshwar Dayal, and Abhay Mehta

Hewlett-Packard Labs
{chetan.gupta,songw,umeshwar.dayal,abhay.mehta}@hp.com

**Abstract.** Given a data set D, such that $(X_i, y_i) \in D, y_i \in \mathbb{R}$, we are interested in first dividing the range of $y_i$, i.e. $(y_{max} - y_{min})$, (where $y_{max}$ is the maximum of all $y_i$ corresponding to $(X_i, y_i) \in D$ and $y_{min}$ is the minimum of all $y_i$ corresponding to $(X_i, y_i) \in D$), into contiguous ranges which can be thought of as classes and then for a new point, $X_j$, predicting which range (class) it falls into. The problem is difficult, because neither the size of each range nor the number of ranges, is known a-priori.

This was a practical problem that arose when we wanted to predict the execution time of a query in a database. For our purposes, an accurate prediction was not required, while a time range was sufficient and the time ranges were unknown a-priori.

To solve this problem we introduce a binary tree structure called *Class Discovery Tree*. We have used this technique successfully for predicting the execution times of a query and this is slated for incorporation into a commercial, enterprise level Database Management System.

In this paper, we discuss our solution and validate it on two more real life data sets. In the first one, we compare our result with a naive approach and in the second, with the published results. In both cases, our approach is superior.

## 1 Introduction

In this work, we address the problem of classification with unknown classes. Given a data set D, such that $(X_i, y_i) \in D$, (where $X_i$ is a vector, corresponding to a multi-dimensional point and $y_i \in \mathbb{R}$), we are interested in first dividing the range of $y_i$, i.e. $(y_{max} - y_{min})$, (where $y_{max}$ is the maximum of all $y_i$ corresponding to $(X_i, y_i) \in D$ and $y_{min}$ is the minimum of all $y_i$ corresponding to $(X_i, y_i) \in D$), into contiguous ranges which can be thought of as classes and then for a new point, $X_j$, predicting which range (class) it falls into.

This problem arose in the context of predicting the execution time of a database query. Any autonomic data warehouse management system needs to have some estimate of the execution time of a query. Previous researchers have focused on predicting precise execution times. In our experience, this is extremely difficult and not reliable even with a moderate accuracy. Furthermore, for our (workload management) purposes, it is actually unnecessary to estimate a precise execution time. It is sufficient to estimate the query execution time in the form of a time range, for instance, to assign queries to different queues. This relaxation allows us to reformulate the problem and bring in the machinery of machine learning to address it. It is precisely this problem of estimating query execution time that motivated this work.

It is desirable for a solution to this problem to have certain properties. We highlight them below, and specify them precisely later in this paper.

1. The number of ranges should be sufficient. It would be meaningless to predict that all queries belong to a single range. That is, it's trivial to predict with 100% accuracy that a new point belongs to the entire range.
2. The span of every divided range should be meaningful. Very small ranges are problematic because of overfitting and very large ranges are not very useful.
3. The user should be able to decide a trade-off between the accuracy of prediction and the number of the ranges according to their requirement.
4. The model for prediction should be cheap to build and deploy.

A question arises as to why such a problem cannot be addressed through traditional regression-based techniques. Certain properties (like execution time of a query) are stochastic and lend themselves to prediction by *least square regression* analysis. In such a scenario, let the response variable (such as query execution time) of the $i^{th}$ object $o_i$, be denoted by $y_i$. Let $X = \{x_{1i}, x_{2i}, \ldots, x_{ki}\}$ be the vector features of object $o_i$. Since the response variable is related to the feature set, the relationship between the response and the features may be mathematically described as $y_i = f(X, \beta_i) + \epsilon_i$, where $\epsilon_i$ $N(0, \sigma^2)$. The best fitting relationship can then be obtained by minimizing the sum of squares of errors given by: $\sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} [y_i - f(X_i, \beta_i)]^2$. This however, requires us to know the function $f$. Very often, the function cannot be obtained. For instance, for query execution time, the interaction between the variables that affect the execution time is not very clearly understood, since the code for the operators can be very complex and not all the variables that affect the query execution time can be measured precisely before or at the time of execution. In other words, at most what we can hope for, is an approximate value or ranges from the function $f$. When this problem is reformulated to discover and predict ranges (instead of precise values), it is expressed as a minimization problem: $F = \sum_{i=1}^{n} [y_i - f_2(f_1(X_i, \beta_i))]^2 + \lambda g(p, w_{max}, w_{min})$. Here, $f_1$ discovers and predicts the range and $f_2$ computes the center of the range as the predicted value for the $X_i$, $p$ is the number of ranges, $w_{max}$ and $w_{min}$ are the sizes of the maximum and the minimum range and $g$ is some function of $p$, $w_{max}$ and $w_{min}$. The function $g$ is expected to satisfy the first two constraints mentioned above. Even this simplified problem (since $f_1$ should be easier to compute than $f$ from the previous equation) is difficult to solve since the function $g$ is unknown.

Hence, we take a data mining approach, where $g$ is enforced algorithmically through constraints and $f_1$ is a data mining classification model instead of a regression function. This overcomes the problem of knowing $g$ algebraically. Finally the problem we solve is to minimize $F = \sum_{i=1}^{n} [\delta_{r(y_i), f_1(X_i)}]$, where $r$ gives the correct range of $y_i$ ($r$ and $f_1$ share the same set of ranges) and $\delta$ is the Kronecker delta function.

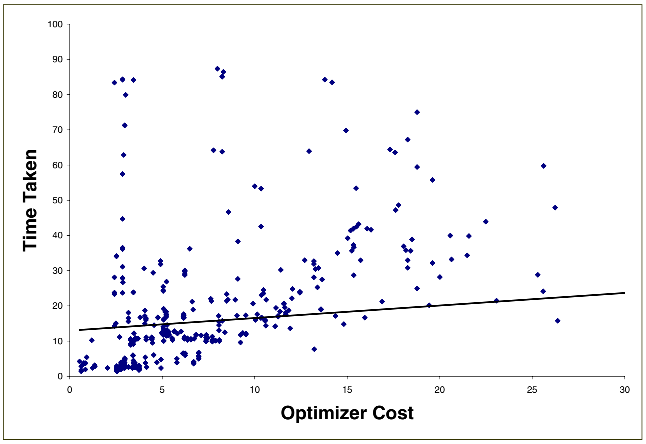


**Fig. 1.** Optimizer Cost and Actual Time Taken

To demonstrate the state of the art, and emphasize our approach, in Figure 1 we have plotted the actual execution time of a set of queries as a function of their estimated optimizer cost from an actual commercial product. Estimated optimizer cost is an analytically computed quantity that a data warehouse optimizer proposes (most commercial products pursue this approach) as an estimate of the actual execution time. It can be easily seen that the proposed cost including the best fit line is a very poor estimate.

There are numerous other problems where classification with unknown classes (or predicting ranges when the ranges are not predetermined) is needed. For example, predicting price categories, or classifying customers based on the total sale value, etc.

To address these problems, we introduce a simple binary tree structure, named as *Class Discovery Tree* or *CDTree*. CDTree can discover the classes during the building phase. CDTree is a binary tree with each node represents a range. Each node contains a binary classifier that divides the corresponding range into two. This way we obtain a set of ranges in the leaf nodes of the CDTree. The CDTree can also be used to solve the classification problem with a large number of classes. Our approach will group the classes into a smaller number of classes and then predict the group class label. We present one such example in the experimental section.

A naive approach to the problem of *Classification with unknown classes* would first use some histogram construction (to partition the whole range into contiguous buckets) or clustering algorithm histogram to create the ranges. And then a multi-class classifier can be learned on the set of ranges. However there are two significant drawbacks in this straightforward approach:

1. In our problem, the number of classes is not known but this is typically required as an input for histogram construction or a clustering algorithm.
2. Histogram methods give a set of contiguous buckets (and this would be true for non-hierarchical clustering like k-means as well). Thus, offering the user

a trade-off between accuracy and the number of ranges would require us to build multiple different classification models.

A possible solution for the second drawback is to use a hierarchical clustering approach to get a nested set of ranges. Later in this paper, we compare this solution with our approach in detail to show why our approach is preferable.

Our contributions in this paper are as follows:

1. We address the problem of first dividing a range into classes and then predicting the class. The classes are not known a-priori.
2. To solve this problem we present a new technique based on a simple data structure called CDTree.
3. We show that CDTree has application for multi-classification problem with a large number of classes.
4. We present a detailed discussion of our methodology.
5. We present results on real life data sets.

In Section 2, we present the related work. In Section 3 our approach is discussed in detail and in Section 4 we compare our approach with the naive approach. In Section 5 we discuss some practical enhancements and in Section 6 the results of our experimental validation are shown. Finally, we conclude this paper in Section 7.

## 2 Related Work

We first discuss the work related to our overall problem. Then, we present some related work on predicting execution times of queries which motivated us.

To the best of our knowledge, there does not seem to be an existing solution to the problem of classification with unknown class. A close technique is Regression Trees [1]. The idea is to fit a regression line to the points in the leaves. This approach differs from ours in several significant ways. First and most importantly, in a regression tree, the leaves need not represent contiguous non-overlapping ranges which are essential to many problems. For example, query scheduling might be (and often is) based on distinct classes of queries defined by the query execution time. Furthermore, regression tree is used for regression and the split decision is made to maximize the decrease in variance, whereas we are interested in classification and our splitting criterion is a maximization of prediction of accuracy for class labels.

As we had discussed earlier, a naive approach would be constructing a histogram followed by a multi-class classification. A good approach to construct a histogram is v-optimal histogram [2] which minimizes the SSE between two distributions. A good discussion on multi-class classification can be found in [3]. We will show through experiments and through discussion that our solution is superior to the naive approach.

CDTree is closely related to the general technique of decision tree (for a good survey see [4]) but differs in various ways, most basic of which is the fact that for our problem the classes are unknown. Another difference with classical decision tree algorithms is that instead of using a single variable to split a node, we use a function (classifier function) of many variables. Tree construction algorithms have considered multivariate splits and one example is a Linear Discriminant Tree [5], which uses a linear machines (linear structures that discriminate between multiple classes) at internal nodes. Again, the class needs to be known for Linear Discriminant Trees.

Researchers and practitioners have built increasingly sophisticated database cost models for query optimization (for example [6]). However, building an accurate analytical model is difficult, especially under varying database load conditions. Using an optimizer's analytical cost model to estimate the actual execution time of a query on a loaded system has met with limited success in the field and it is common knowledge that query cost estimates produced by query optimizers do not accurately reflect query run times (Please refer to Figure 1). Analytical approaches have been used for estimating query response times [7,8] and there are a few commercial products that use analytical and simulation models to predict query execution times [9,10,11]. The analytical approaches depend on the creation of resource models which are notoriously complex and difficult. Assumptions that have to be made (for example, exponential service time distributions) may not be true in practice, and hence the results may not be relevant.

Classification with unknown classes is a difficult problem. An earlier version of CDTree is used specifically for prediction execution times [12]. In this paper we abstract the problem to the problem of unknown classes and present a thorough solution and detailed discussion.

## 3 Our Solution - CDTree

We state some definitions for the purpose of clarity in the text that follows. First we specify the range of a set of points. The range of a set of points is the span of the $y$ values:

**Definition 1.** *The range of a set $S \subset D$ of points $(X_i, y_i) \in S$ is $[y_{min}, y_{max}]$, where $y_{max}$ is the maximum value and $y_{min}$ is the minimum value for all $y_i$, and $X_i$ corresponds to the vector forming the point.*

The predicted range or the class of a point is essentially the range that the point is predicted to fall into:

**Definition 2.** *The class or the range of a point $(X_i, y_i) \in D$ is the range $[y_a, y_b)$ or $[y_a, y_b]$, where $y_i$ lies.*

We now precisely define a CDTree.

**Definition 3.** *A CDTree, denoted by $T_s$, is a binary tree such that:*

1. *For every node (or leaf) $u$ of $T_s$ there is a range associated $[u_a, u_b)$. (For the right-most node the range is right closed as $[u_a, u_b]$. This will not be repeated below.)*

2. *For every node $u$ of $T_s$, there is an associated 2-class classifier $f_u$.*
3. *The node $u$ contains examples $E_u$, on which the classifier $f_u$ is trained.*
4. *The node $u$ contains examples $V_u$ which are used for validation*
5. *$f_u$ is a classifier that decides for each new point $i$ if $i$ should go to $[u_a, u_a+\Delta)$ or $[u_a+\Delta, u_b)$, which are the ranges of $u$'s children.*
6. *For every node $u$ of $T_s$, there is an associated accuracy $A_u$, which is measured as the percentage of correct predictions made by $f_u$ on the validation set $V_u$.*

CDTree is a binary tree, where every node of the tree represents a range and the children's ranges are non-overlapping sub-ranges that cover the parent's range. These ranges form a tree of nested ranges. Every node contains an example set $E_u$ and a validation set $V_u$. The $y$ values of points in $E_u$ and in $V_u$ fall in the range of the node $u$. Conversely, from all the examples in the training set, the points whose $y$ value falls in the range of node $u$ are in $E_u$ and from all the examples in the validation set, the points whose $y$ value falls in the range of node $u$ are in $V_u$.

While building this tree, at every node we not only need to find the two sub ranges for the range of that node but also a classifier that can predict the two ranges, i.e., in a CDTree, at each node we need a combination of two meaningful classes and a classifier.

We fix a set of classifiers $F$ as candidates. For example, the set can include three classifiers: nearest neighbor, C4.5 and Logistic Regression. For every node, we compute a set of possible separation points $S$ by looking at the points in the example set $E_u$. For each $f \in F$, and each $s \in S$ we build a classifier on the example set $E_u$. Then, from these combinations of the classifiers and the ranges, we choose the best one with the highest accuracy on the validation set $V_u$. We discuss this in further detail in the next section.

When a new data point $X_i$ comes in, it traverses down the constructed CDTree in a top-down manner to a leaf $l$. The range of the leaf $l$ is the predicted range for the point $X_i$. In fact, the user can choose any range that lie on the path from the root to the leaf $l$.

*Example 1.* In Figure 2, we present a sample CDTree. This was obtained in one of our preliminary experiments where we were trying to predict the execution time of a database query. The classifier, $f_u$ associated with the root node is a classification tree with a time range of $[1, 2690]$ seconds. The root node has two children that divide its range into two: $[1, 170)$ and $[170, 2690]$. The associated accuracy of this classifier is 93.5%, i.e., for 93.5% of the example queries in the validation set $V_u$ of the root node, the classifier was able to predict the query's time range correctly. The rest of the nodes can be understood similarly. It can be seen that the CDTree has five leaves. The final time ranges in second are:

$$\{[1, 13), [13, 170), [170, 1000), [1000, 1629), [1629, 2690]\}$$

The skew towards the smaller ranges correctly reflect the skew of the underlying distribution of query execution times. Also note that, the CDTree is a hierarchical structure and the ranges are hierarchical, i.e., the children's range is nested in the parent's range.

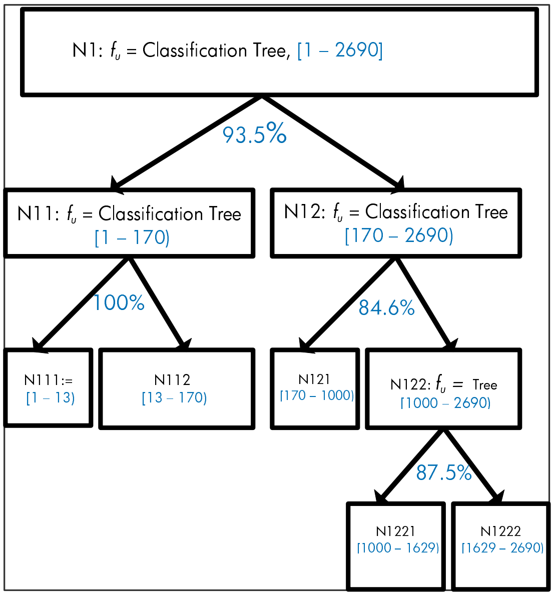

N1: $f_v$ = Classification Tree, [1 – 2690]
93.5%
N11: $f_v$ = Classification Tree [1 – 170)
N12: $f_v$ = Classification Tree [170 – 2690)
100%
84.6%
N111:= [1 – 13)
N112 [13 – 170)
N121 [170 – 1000)
N122: $f_v$ = Tree [1000 – 2690)
87.5%
N1221 [1000 – 1629)
N1222 [1629 – 2690)

**Fig. 2.** Sample CDTree

### 3.1 Building a CDTree

We now present a procedure for building a CDTree. Like all tree constructions, a CDTree is built recursively. This is done by splitting the range of the parent node, till some stopping criterion is reached. That is:

1. Start with all the training examples in the root node.
2. Find a point $X_s$ such that corresponding $y_s$ lies within the range of the node.
3. Split the node into two, such that all points with $y_i < y_s$ go into the left node and all points with $y_i \geq y_s$ go to the right node.
4. Recursively keep splitting the nodes till a termination condition is reached.

**Termination Conditions.** We want our tree to have meaningful ranges. If the range for a node is too small no additional useful information might be gained by further splitting it. Besides meaningful ranges, we also want to have meaningful classes. That is a class should contain at least a minimum number of points. This is important since we are discovering classes and it does not make much sense to let a class have very few points in it. Thus, we specify our stopping criterion as follows:

1. *Minimum Interval Size* When the range of a node falls below a threshold do not further split that node. We call this threshold $min_{IntervalSize}$.
2. *Minimum Number of Examples* When the number of examples in a node falls below a threshold do not further split the node. We call this threshold $min_{Example}$

When we use only these two termination conditions we call it a *Complete CDTree.*

We now specify how to build the node.

**Node Construction.** We now describe how an individual node is constructed:

1. For a node $u$, take all the points $(X_i, y_i)$ in the training example set $E_u$ and arrange them in ascending order of $y_i$.
2. The average of each successive pairs of $y_i$ is one possible separation point (class boundary). All possible separation points form a set $S$.
3. To avoid overfitting, we enforce two conditions mentioned earlier. (1) If, as result of choosing a possible separation point, one of the children contains fewer than $\frac{(min_{Example})}{2}$ points, that separation point is removed from the candidate set $S$. (2) Similarly, if the range of one of the children is less than $\frac{(min_{IntervalSize})}{2}$, then that separation point is invalid. Let $S'$ be the new set of separation points after applying the two conditions.
4. For each $f \in F$, and for each $s \in S'$, build a classifier on $E_u$ that classifies all $X_i \in E_u$ into two sub-ranges, in one $y_i < s$ and in the other $y_i \geq s$. This step gives us a several pairs of classification functions and separation points $(f, s)$.
5. For each pair $(f, s)$, we compute the accuracy of predicting the two classes on the validation set $V_u$, i.e., we divide $V_u$ based on $s$ and compute the accuracy of predicting the classes using $f$.
6. Choose the pair $(f, s)$ that has the highest accuracy as $f_u$ for the node $u$.

*Example 2.* Let's say that node $u$ consists of 10 points where the corresponding $y$ values are $\{16, 2, 5, 9, 5, 17, 3, 14, 2, 3\}$. Let $F$ be {1-NN, C4.5}, $min_{IntervalSize} = 4$ and $min_{Example} = 6$. We first sort $y$ in ascending order to get: $\{2, 2, 3, 3, 5, 5, 9, 14, 16, 17\}$. Now we compute the set of possible separation points and obtain $S = \{2, 2.5, 3, 4, 5, 7, 11.5, 15, 16.5\}$. For separation points $\{2, 2.5, 3, 4\}$ the range value of the children will be less than $\frac{4}{2} = 2$. This is also true for $\{16.5\}$. So these points are removed from $S$. For separation points $\{2, 2.5, 3, 4\}$ and $\{15, 16.5\}$ the number of examples in the children would be less than $\frac{6}{2} = 3$. After removing these invalid separation points we get $S' = \{5, 7, 11.5\}$. We then consider each of these separation point in turn. Suppose that, when we split $V_u$ on 5, for the two classifiers we get the accuracy as 70% and 72%, respectively; similarly, when we split $V_u$ on 7, for the two classifiers we get the accuracy as, 75% and 73%, respectively; and when we split $V_u$ on 11.5, for the two classifiers we get the accuracy as, 67% and 68%, respectively. Then, for this node the best separation point is 7 and the classifier, $f_u$ is 1-NN, since that combination of accuracy and separation point has the highest accuracy of all the combinations.

**Avoiding Overfitting.** In classical decision trees, overfitting occurs when nodes continue to be split in order to achieve "pure" nodes. This can lead to nodes having a very few points. Similarly, in our problem when we discover the two classes, if the number of examples from one class is far fewer than the number of examples in the second class it can lead to overfitting, i.e., the small class could exist only in the examples on which the classifier was created. To avoid that we add a condition on the creation of possible separation points:

1. *Skipping Separation Points.* Do not consider a separation point $s$ if as a result of separation at $s$, one of the children contains less than $n_{skip}$-percent of the total number of examples in $E_u$.

Another way of looking at it is that while doing exhaustive search, skip $n_{skip}$ percentage of separation points from the beginning and the end of range.

## 3.2 Discussion

We now discuss our choice of splitting rule, pruning. First we highlight two properties of CDTree:

1. In a CDTree, the classes are nested, i.e., the children's ranges are non-overlapping subsets of the range of the parent node. This offers user a tradeoff between the number of ranges and the accuracy.
2. In a CDTree, the accuracy monotonically decreases as we travel down the tree.

The second property is undesirable. This is because errors are *additive.* For example, suppose $m_1$ points are misclassified at the root node, they will be misclassified all the way to the leaf. Now assume that $m_{2a}$ and $m_{2b}$ points are misclassified at the left child of the root and the right child of the root respectively, then at depth two the total numbers of errors is $m_1 + m_{2a} + m_{2b}$.

**Splitting Rule.** In traditional classification trees, the largest reduction in entropy or the largest information gain is used to measure the performance. Accuracy is not traditionally used because:

1. Accuracy does not reflect the potential for finding a better splitting point.
2. The change in accuracy might be zero for all splits.

In the case of classical classification trees, the points have labels a-priori. So even if a point is mislabeled at the parent node, it could still be correctly labeled at a subsequent node. In our case since labels do not exist *a-priori*, once a point is incorrectly classified it can never be correctly classified again. So it makes more sense to locally optimize for a maximal gain in accuracy. Furthermore, computing information gain (or entropy) while dividing the range is not a natural definition of entropy. For example, in Regression Trees [1] which address a problem similar to ours, the split that results in the largest reduction in variance is chosen.

**Pruning.** In Minimum Cost-Complexity Pruning [1], a set of trees of decreasing size (and contained in each other) are obtained. From this set, a final tree is chosen based on the accuracy on the validation set. In our case, since classes are continually created as the tree grows, to avoid overfitting we can:

1. Choose the classes based on the validation set: This step is built into the construction of the tree.
2. Restrict the splitting of a node: For this we introduce the improvement discussed earlier, wherein if a class were to contain a few points, it is not created. Allowing small classes could lead to overfitting because the small class could exist only in the examples on which the classifier was created. This can be thought of as a form of *Pessimistic Pruning.* We will show in our results that this leads to an improvement in accuracy.

## 4 Comparison with the Naive Approach

Finally, we revisit the initial discussion on how our approach is superior to the naive approach. As we mentioned earlier, a naive approach to solving our problem could be:

1. First, create the ranges using some form of clustering or histogram construction, where a range is partitioned into contiguous buckets.
2. Then, learn a multi-class classifier on this set of ranges.

There are significant problems with this straightforward approach:

1. The number of classes is not known which is typically the input required for constructing a histogram or for clustering. For CDTree, we discover the classes as we build the tree. This eliminates the need for knowing the number of classes *a-priori.*
2. Histograms give a set of contiguous buckets. To offer the user a choice at the time of prediction, in a tradeoff between accuracy and the number of buckets, we would have to build different classification models for each number of buckets. On the other hand, in a CDTree we obtain a set of nested ranges. Thus in a single CDTree, a user can choose to stop at an intermediate node instead of traveling all the way down to a leaf, if the result ranges are sufficient. This will give a result with a higher accuracy. For example consider the tree shown in Figure 2, if a user is satisfied with the range of $[1000, 2690]$, she need not further decide if the point lies in $[1000, 1629)$ or $[1629, 2690]$.
3. In our initial experiments, we used some histogram techniques (such as equi-width, equi-depth) and then used multi-class classifiers. The results were consistently inferior. Clustering gives better results than histogram as the first step. In the experimental section we compare with the clustering approach and show that our results are better than those obtained with clustering.
4. A way of obtaining partitions might be to cluster the points. All clustering algorithms suffer from the first problem, i.e., the number of clusters need to be known *apriori.* Partitional clustering (such as, k-means) suffers from the second problem too since it cannot offer the user a tradeoff between accuracy and the number of ranges. Furthermore, clustering under constraints (in our case minimum size of the range, minimum number of examples in a class) is not a very well understood problem.

5. A solution for the problem of tradeoff is to get a nested set of ranges and thus a hierarchical clustering approach can be used. Any solution that could be given by hierarchical clustering is searched in our exhaustive search, i.e., all possible permissible separation points are considered. Surely, they would contain any partition suggested by a hierarchical clustering algorithm.
6. As opposed to combining several binary classifiers, which happens in our approach, a multi-class classifier considers all the classes at once, and this makes the overall optimization problem more expensive to solve [13].
7. We have observed in our experiments that different regions of $y$ have different rules for classification. For example, when predicting database query execution times, smaller queries were classified on the optimizer cost and the larger queries by the load on the system. A binary classifier not only discovers these rules but also helps make sense of the classification itself, which is the essence of a good data mining approach. (This could also be obtained with a multi-class approach where different classes are grouped together, but that approach is not as accurate as one-against-all approach for multi-class classification.)
8. Finally, we demonstrate in our experimental section that our approach gives consistently better results than the naive approach.

## 5 Practical Enhancements

In this section we suggest two practical enhancements over a *Complete CDTree*.

### 5.1 Bounding the Accuracy

We introduce a new termination condition on the accuracy:

1. *Minimum Accuracy.* Do not split a node if the highest accuracy on the validation set is below a threshold $min_{accuracy}$.

As a result of terminating early because of accuracy, the tree does not grow unnecessarily deep. Note that, this does not result in an increase in accuracy by itself because such a tree will be contained in the complete CDTree. (Too deep trees can cause overfitting in classical decision trees, but as we have discussed earlier this is not a primary reason for overfitting in CDTrees).

Fixing $min_{accuracy}$ is a way to decide the tradeoff between the number of ranges and the overall accuracy of the tree. In our experiments below, we use $min_{accuracy} = 0.8$. There is no theoretical reason for choosing this number. This could be an interesting problem for further study but in our experience with prediction of query execution times, 80% accuracy does lead to trees with ranges and accuracy that meet user expectations. Another advantage of early termination is that the tree is easier to comprehend in terms of simpler classification rules.

## 5.2 Reducing the Computational Complexity

Exhaustive search for the best separation point can be expensive. For every separation point in the set $S'$, all the classifiers in the set $F$ are trained and validated. To reduce the computation of exhaustive search, we could look at a subset of the candidate separation points $S'$. To do that we first compute the *gaps*. As before, first sort the $y_i$ in ascending order such that $y_{i+1} \geq y_i$. And then compute the gaps, $\delta$ as: $\delta(y_{i+1}) = \frac{y_{i+1}-y_i}{y_i}$.

1. *Largest Gaps.* Take the $y_i$ corresponding to the largest $n_{gap}$ values as the possible separation points.

*Example 3.* Let's say node $u$ consists of 10 points where there $y$ values are $\{16, 2, 5, 9, 5, 17, 3, 14, 2, 4\}$. Let $n_{gap} = 2$. We first sort $y$ in ascending order to get: $\{2, 2, 3, 4, 5, 5, 9, 14, 16, 17\}$. The gaps are {0,0.5,0.33,0.25,0,0.8,0.55,0.14,0.06}. Then the largest $n_{gap}$ differences are $0.8, 0.55$. They correspond to the splitting points $(9, 14)$.

Note again that doing this does not necessarily increase the accuracy and has the potential to decrease it, as compared to the complete CDTree construction. This is because the set of partitions obtained in this way is a subset of possible separation points obtained through exhaustive search. We suggest $n_{gap} = 5$. Again, there is no theoretical reason, but this gave us the speedups we were looking for in a large number of experiments.

For example, given a dataset containing 473 training examples, 142 validation examples and 154 testing examples, with 15 attributes, without $n_{gap}$ it takes 13913 ms to construct and test a CDTree, and with $n_{gap} = 5$ it takes 3857 ms, a speedup of about 3.6.

## 6 Experimental Results

We present results over three different data sets. As we had mentioned earlier, this work arose out of a practical consideration, namely predicting the execution times of database queries. We will present a set of results regarding this data set. We have also used two other data sets to show the effectiveness of our approach. The first data set, Abalone data set, has a large number of classes (29). We show CDTree groups these classes into smaller number of classes. We compare our results with a naive approach and the published results. For the second data set on housing data in Boston, we compare our results with the naive approach.

For all the experiments the set $F$ of classifier functions was 1-nearest neighbor and decision trees with four different settings. We have used Weka's [14] publicly available libraries. An important point to note is that for the purpose of these experiments we have used a small set for $F$. In real life applications, $F$ could be larger, possibly leading to higher accuracies.

In our results, we report the number of ranges and the accuracy. In order to make a fair comparison between our CDTree approach and the naive approach, the number of ranges is set to be the *same*. This is achieved by using the number of ranges obtained using the CDTree approach as input for the number of clusters for the naive approach. Hence, when comparing the results of the two approaches, it should be kept in mind that the naive approach is essentially solving only half the problem.

Typically as the number of ranges goes up, the accuracy goes down (This can be clearly seen in Table 2). Without the creation of some artificial metric it is not possible to say which tree is better, one with higher accuracy and fewer ranges or the one with lower accuracy but with the more ranges. These are tradeoffs and we would prefer "high" accuracy and "reasonable" number of ranges. As mentioned before, in our experiments below we use $min_{accuracy} = 0.8$ to achieve this tradeoff.

In general, since we were motivated with a real life problem, we left the decision on the tradeoff to the user. Here, the property of the CDTree that the classes are nested and a user can choose to stop at a intermediate node without going to a leaf node is useful for the user.

For the experiments, we construct CDTrees starting from a complete tree and then adding on the improvements, to demonstrate how the results are affected with the improvements. However, we first discuss how we constructed the naive approach.

## 6.1 Naive Approach

Recall, that the naive approach is a two step process:

1. Cluster the data set on the $y$-values.
2. Fit a multi-class classifier on this data.

The ranges obtained as a result of clustering should follow the constraints of $min_{example}$ and $min_{size}$. There has been some work on the clustering with constraints but there is no standard approach. We construct a basic algorithm to do this. Note that clustering requires us to know the number of clusters. Let the number of clusters be $n_c$.

1. Using simple $k$-means find the $n_c$ clusters, where the $k$ value is given by the number of ranges obtained from corresponding CDTree approach.
2. If all the clusters meet the $min_{example}$ and $min_{size}$ constraints, stop. Otherwise increase the number of clusters by 1 in k-means and goto step 1.
3. Assign the points of all the clusters that do not meet the criterion to the cluster with the centroid nearest to them in terms of Euclidean Distance.
4. Count the number of clusters, if it is $n_c$ then stop. Otherwise increase the number of clusters by 1 in k-means and goto step 1.

Once we have this clustering, we use clusters as class labels. For a fair comparison, we take all $f \in F$ and make multi-class classifiers with 1-against all (which is known to have higher accuracy) with each $f$. Then for each multi-class classifier we compute the accuracy on the test set. We finally report the highest accuracy amongst all these classifiers.

## 6.2 Predicting Execution Times for Queries

This set of experiments for Customer X (for confidentiality reason we cannot disclose the name of the customer) was run on a database that was installed on a machine with 256 Intel Itanium processors. We took a set of actual BI (Business Intelligence) queries run in a single day by one of our customers. They include both ad-hoc queries and canned reports. There were a total of 769 queries in this data set. We created 12 different workloads by running a different number of queries together. For each query, in each workload, we collected the execution times. The $X$ values for each query are certain properties of the query and the load on the system, the $y$ values are the execution times. This way, each workload gives us a data set. As mentioned before, the problem of predicting execution times of a database query is difficult. For more detailed results please refer to our previous work [12] where an earlier version of CDTree was used.

**Complete CDTree.** For each of the twelve data sets (where a data set corresponds to a workload), we create 10 different test sets by randomly distributing 60% of the points as the training set, 20% of the points as the test set and 20% of the points as the validation set. First, we present the results for a complete CDTree. We have tabulated the averages for the ten runs per data set and compared our results with the average of the naive approach. The results were obtained with with $min_{IntervalSize} = 1$ and $min_{Examples} = 25$. They have been tabulated in Table 1.

Each row of Table 1 represents one of the 12 workloads used in the experiment. The second column shows the average accuracy obtained from the 10 test sets using CDTree. The third column shows the average accuracy using the naive approach with the same number of ranges as the CDTree. The number of ranges obtained with CDTree is is shown in the fourth column. (For the naive approach, for each run, the number of ranges obtained from CDTree was input as the

**Table 1.** Results Obtained with a Complete CDTree

| Data Set | Accuracy%(Complete CDTree) | Accuracy%(Naive) | # of Ranges |
|---|---|---|---|
| 1 | 79.87 | 65.73 | 9.6 |
| 2 | 77.71 | 74.05 | 10.1 |
| 3 | 72.16 | 64.57 | 10.2 |
| 4 | 66.01 | 51.84 | 11.2 |
| 5 | 53.20 | 50.38 | 13.2 |
| 6 | 68.44 | 62.97 | 12.9 |
| 7 | 66.41 | 56.68 | 13.5 |
| 8 | 72.45 | 62.09 | 13.4 |
| 9 | 65.22 | 56.98 | 14.2 |
| 10 | 70.93 | 63.08 | 12.1 |
| 11 | 73.38 | 66.69 | 11.3 |
| 12 | 69.87 | 57.28 | 12.9 |

number of clusters.) It can be seen that our approach outperforms the naive approach for all twelve data sets. Also, we can observe that in general, the accuracy of CDTree decreases with an increase in the number of ranges.

**Using $min_{accuracy}$.** Next, we present the results when we introduce the minimum accuracy criterion. We set $min_{accuracy} = 0.80$. The rest of the parameters are the same as those for the complete CDTree discussed above. The results have been tabulated in Table 2. The result can be interpreted in the same way as Table 1. It can be seen that CDTree again outperforms the naive approach. In comparison with Table 1, it can be observed that the effect of using $min_{accuracy}$ with CDTree, is an increase in accuracy and a decrease in the number of ranges. This is the desired effect of introducing the minimum accuracy criterion.

**Table 2.** Results Obtained with a CDTree with $min_{Accuracy} = 80$

| Data Set | Accuracy%(CDTree with Min Accuracy) | Accuracy%(Naive) | # of Ranges |
|---|---|---|---|
| 1 | 95.43 | 89.76 | 3.6 |
| 2 | 89.81 | 84.11 | 4.8 |
| 3 | 81.67 | 83.76 | 5.4 |
| 4 | 72.67 | 67.25 | 9.4 |
| 5 | 82.67 | 79.23 | 7.7 |
| 6 | 69.30 | 65.59 | 12.3 |
| 7 | 70.81 | 66.08 | 11.9 |
| 8 | 76.70 | 67.13 | 11.7 |
| 9 | 72.86 | 64.10 | 12.3 |
| 10 | 72.46 | 66.80 | 11.6 |
| 11 | 76.59 | 74.84 | 9.9 |
| 12 | 72.98 | 64.28 | 12.0 |

**With $min_{accuracy}$ and $n_{skip}$.** Now we present the results of CDTree with $min_{accuracy}$ and $n_{skip}$. We set $n_{skip} = 0.25$, i.e., skip 25% points as possible separation points from the beginning and the end of a range. The rest of the parameters are the same as before. We have plotted the results for all 120 experiments (12 workloads * 10 test settings) in Figure 3, where the accuracy of the prediction is the x-axis and the number of ranges is the y-axis. For some of the predictions the accuracy reaches 100% and the accuracy is 90% and above for a large number of experiments.

Note that setting $n_{skip}$ is necessary even when $min_{accuracy}$ is used, since $min_{accuracy}$ specifies the lower bound of accuracy of the classification while $n_{skip}$ defines the minimum size of ranges.

### 6.3 Multi-class Classification with Large Number of Classes

The CDTree approach can be used in a multi-class classification problem where there are a large number of classes. CDTree will automatically group these classes into a smaller number of classes.

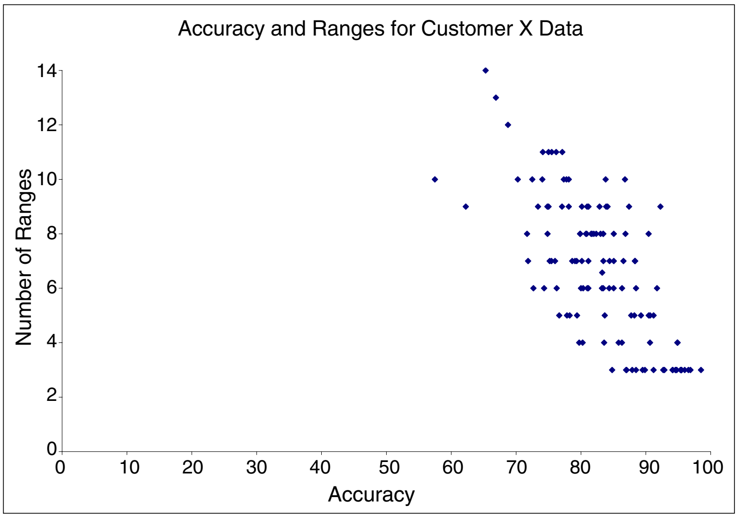


**Fig. 3.** Query Execution Times Prediction Results for Customer X

To demonstrate this, we use the Abalone (a type of shell fish) data set which predicts the age of abalone from physical measurements. The attributes are categorical, real and integer. The number of instances is 4177. The number of classes is 29. This is pretty difficult data set to classify. The accuracy on the data set is known to be 72.8% [15] when the classes were grouped into three classes *apriori* by the researchers themselves.

The results of for this problem were obtained with the following settings: $min_{IntervalSize} = 3, 4, 5$ and $min_{Examples} = 25$, $min_{accuracy} = 0.80$ and $n_{skip} = 0.25$. Like the previous experiments, we used 60% of the data for training, 20% for validation and the last 20% for testing. Again, as before, we created 10 runs and report the averages.

The results are presented in Table 3. It can be observed that with all three of our approaches we get high accuracy and a larger number of ranges (group of classes) as compared to the naive approach. For the CDTree with $min_{accuracy} + n_{skip}$, not only is the accuracy higher than the best known result (to the best of our knowledge) but the number of classes is also higher.

Most importantly, CDTree discovered the class groups on its own. In the previous work, it was not clear how the authors arrived at their grouping. In our case, these were discovered by the CDTree and they closely matched with groupings used in the previous work.

### 6.4 Data Set 2: Boston Housing Data

This is a data set of housing prices in Boston. We have obtained it from the UCI repository [16]. It has 14 attributes of real and integer types. There are 506 instances. This data set is typically used for regression. Like the previous experiments, we used 60% of the data for training, 20% for validation and the

**Table 3.** Abalone Data Set Results

| Method | CDTree(%) | Naive(%) | #_Ranges |
|---|---|---|---|
| Complete Tree | 69.57 | 45.86 | 6.0 |
| $min_{accuracy}$ | 70.93 | 52.08 | 5.2 |
| $min_{acc.} + n_{skip}$ | 73.63 | 60.06 | 4.2 |

**Table 4.** Boston Housing Data Set Results

| Method | CDTree(%) | Naive(%) | #_Ranges |
|---|---|---|---|
| Complete Tree | 60.9 | 57.43 | 6.5 |
| $min_{accuracy}$ | 72.24 | 66.63 | 5.3 |
| $min_{acc.} + n_{skip}$ | 72.49 | 66.65 | 5.2 |

last 20% for testing. As before, we created 10 runs and report the averages. The results for complete CDTree were obtained with $min_{IntervalSize} = 10$ and $min_{Examples} = 25$. When $min_{accuracy}$ was added it was as set to 0.80 and $n_{skip}$ was added it was set to 0.25.

The results have been presented in Table 4. As before, there is an increase in overall accuracy at the expense of ranges as we move away from the Complete CDTree. However, there is no significant improvement with addition of $n_{skip}$. This could be because no classes could be found in the part of ranges that was to be skipped, which had accuracy greater than $min_{accuracy}$. Some of the sample ranges were:

1. With five ranges: $\{[5.0, 12.6), [12.6, 25.1), [25.1, 31.5), [31.5, 37.2), [37.2, 50.0]\}$. These were obtained without $n_{skip}$
2. With six ranges: $\{[5.0, 12.6), [12.6, 17.8), [17.8, 25.1), [25.1, 31.5), [31.5, 37.2), [37.2, 50.0]\}$. These were obtained with $n_{skip}$.

## 7 Conclusions

In this work, we address the problem of classification with unknown classes using a simple binary tree structure *CDTree*. Another application of CDTree is a classification problem with a large number of classes. When building the tree, certain properties were desired: (i) the ranges should be sufficient in number; (ii) the span of a range should be meaningful; (iii) the user should be able to choose a tradeoff between the accuracy of prediction and the number of ranges; (iv) the model for prediction should be cheap to build and deploy. Our CDTree achieves all these goals. We also presented a detailed discussion of our methodology and demonstrated the effectiveness of our approach on real life data sets. We have used this technique successfully for predicting the execution times of a query and this is slated for incorporation into a commercial, enterprise class Database Management System.

## References

1. Brieman, L., Friedman, J.H., Olsen, R.A., Stone, C.J.: Classification and Regression Trees. Chapman Hall/CRC (1984)
2. Jagadish, H.V., Koudas, N., Muthukrishnan, S., Poosala, V., Sevcik, K.C., Suel, T.: Optimal histograms with quality guarantees. In: VLDB, pp. 275–286 (1998)
3. Allwein, E., Schapire, R.E., Sanger, Y.: Reducing multiclass to binary: a unifying approach for margin classifiers. Journal of machine learning research 1, 113 (2001)

4. Murthy, S.K.: Automatic construction of decision trees from data: A multi-disciplinary survey. Data Mining and Knowledge Discovery 2(4), 345–389 (1998)
5. Brodley, C.E., Utgoff, P.E.: Multivariate decision trees. Machine Learning 19, 45–77 (1995)
6. Graefe, G.: Query Evaluation Techniques for Large Databases. ACM Comput. Surv. 25(2), 73–170 (1993)
7. Salza, S., Renzetti, M.: Performance Modeling of Paralled Database System. Informatica (Slovenia) 22(2) (1998)
8. Tomov, N., Dempster, E.W., Williams, M.H., Burger, A., Taylor, H., King, P.J.B., Broughton, P.: Analytical response time estimation in parallel relational database systems. Parallel Computing 30(2), 249–283 (2004)
9. Eberhard, R.: DB2 Estimator for Windows, IBM (1999), http://www.software.ibm.com/data/db2/os390/estimate
10. Garth, M.: Modelling parallel architectures, Metron Tech. White Paper (1996), http://www.metron.co.uk/papers.htm
11. Platinum Technology: Proactive performance engineering, Platinum Technology White paper (1999), http://www.softool.com/products/ppewhite.htm
12. Gupta, C., Mehta, A., Dayal, U.: PQR: Predicting Query Execution Times for Autonomous Workload Management. In: Proc. of International Conference on Autonomic Computing (2008)
13. Hsu, C.W., Lin, C.J.: A comparison of methods for multi-class support vector machines. IEEE Trans. on Neural Networks 13, 415–425 (2002)
14. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, San Francisco (2005)
15. Abdelbar, A.M.: Achieving superior generalisation with a high order neural network. Journal Neural Computing & Applications 7(2), 141–146 (1998)
16. Asuncion, A., Newman, D.: UCI machine learning repository (2007)

# HSM: Heterogeneous Subspace Mining in High Dimensional Data

Emmanuel Müller[1], Ira Assent[2], and Thomas Seidl[1]

[1] Data management and exploration group, RWTH Aachen University, Germany
{mueller,seidl}@cs.rwth-aachen.de
[2] Department of Computer Science, Aalborg University, Denmark
ira@cs.aau.dk

**Abstract.** Heterogeneous data, i.e. data with both categorical and continuous values, is common in many databases. However, most data mining algorithms assume either continuous or categorical attributes, but not both. In high dimensional data, phenomena due to the "curse of dimensionality" pose additional challenges. Usually, due to locally varying relevance of attributes, patterns do not show across the full set of attributes.

In this paper we propose HSM, which defines a new pattern model for heterogeneous high dimensional data. It allows data mining in arbitrary subsets of the attributes that are relevant for the respective patterns. Based on this model we propose an efficient algorithm, which is aware of the heterogeneity of the attributes. We extend an indexing structure for continuous attributes such that HSM indexing adapts to different attribute types. In our experiments we show that HSM efficiently mines patterns in arbitrary subspaces of heterogeneous high dimensional data.

## 1 Introduction

Data mining is a knowledge discovery task that aims at identifying interesting novel patterns in the data. Several subdisciplines exist, that typically focus on different types of application scenarios [1]. In unsupervised learning, clustering and frequent itemset mining are widely studied and applied. Whereas clustering aims at automatically grouping similar objects into groups, separating dissimilar ones, frequent itemset mining tries to detect objects that occur repeatedly in the data. Clustering algorithms for continuous and categorical data sets have been proposed in the literature [2,3], whereas frequent itemset mining assumes categorical data. Furthermore, there are new challenges associated with the "curse of dimensionality" in high dimensional data, i.e. data with very many attributes [4]. As a consequence, research has focused on detecting patterns in projections of the full dimensional space [5,6,7].

In this work, we focus on mining of heterogeneous data types with continuous and categorical attributes, as found in many practical applications. The discovery of meaningful patterns that provide a consistent view on both data types requires a pattern model that is consistent in defining interestingness and novelty for continuous and categorical attributes alike.

We propose a novel model for heterogeneous data mining in high dimensional data. Our model provides a consistent view on both data types, and derives a common subspace clustering notion. Based on previous work on density-based subspace clustering in continuous data [7], we extend our model in this work to heterogeneous data. We highlight common challenges in subspace clustering and frequent itemset mining by giving a thorough review and comparison of models in both areas, before we present our novel model covering both areas. Our consistent model ensures comparable subspace clustering results on heterogeneous data by a consistent normalization of both density and frequency measures.

As recent research has shown, pattern detection in projections of high dimensional data is a computationally challenging task [5,6,7]. Especially for different data types, we need adaptive algorithms which ensure efficient pattern detection in projections of heterogeneous data. We propose an algorithm for heterogeneous subspace clustering that exploits the nature of continuous and categorical attributes in a novel index structure. This index is based on our previous work on efficient density-based subspace clustering for continuous data [8]. We extend this SCY-tree for continuous attributes in this work to an adaptive HSM-tree index structure for heterogeneous data, by extracting common mining characteristics of different attribute types in subspace clustering and frequent itemset mining. Although it seems to be a simple combination of known indexing techniques, the novel automatic adaptation between continuous and categorical attributes is essential for efficient mining on heterogeneous data. As we show in thorough evaluations our algorithm achieves a significant efficiency improvement compared to recent subspace clustering algorithms.

As a running example to illustrate our work, we use the toy data set given in Figure 1. Our example is based on a real world scenario: A sensor network measuring different types of environmental parameters, vegetation and prevailing animal types around the sensor. We have given a data set of 18 objects (sensor measurements) described by 6 dimensions (sensor types). There are four continuous data types: noise level, temperature, humidity, and illumination, as well as two categorical data types: vegetation and animals. We have highlighted the hidden patterns in the data by surrounding lines. For example we see that there are three frequent itemsets present in the categorical attributes. Each of them specifies a group of objects: $o_1 - o_7$, $o_8 - o_{12}$ and $o_{13} - o_{18}$. Taking also the continuous valued attributes into account one can observe the same grouping for the humidity attribute, while noise and temperature form other groupings. However, groups do not appear in all attributes, as the attribute illumination is noisy and provides almost random measurements. The goal is thus to identify a grouping of objects that are similar in a subspace of the high dimensional space, i.e. to identify the relevant subset of dimensions on which the grouping appears.

| | $d_1$ noise level | $d_2$ temperature | $d_3$ humidity | $d_4$ illumination | $d_5$ vegetation | $d_6$ animals |
|---|---|---|---|---|---|---|
| $o_1$ | 3 dB | 15°C | 54% | 20000 lx | FOREST | MAMMALS |
| $o_2$ | 7 dB | 13°C | 53% | 70000 lx | FOREST | MAMMALS |
| $o_3$ | 5 dB | 12°C | 58% | 100 lx | FOREST | MAMMALS |
| $o_4$ | 7 dB | 10°C | 56% | 80 lx | FOREST | MAMMALS |
| $o_5$ | 8 dB | 9°C | 54% | 20 lx | FOREST | MAMMALS |
| $o_6$ | 6 dB | 9°C | 52% | 1 lx | FOREST | MAMMALS |
| $o_7$ | 3 dB | 8°C | 53% | 0,0001 lx | FOREST | MAMMALS |
| $o_8$ | 4 dB | 12°C | 17% | 15000 lx | GRASSLAND | INSECTS |
| $o_9$ | 5 dB | 11°C | 13% | 60000 lx | GRASSLAND | INSECTS |
| $o_{10}$ | 5 dB | 10°C | 15% | 10 lx | GRASSLAND | INSECTS |
| $o_{11}$ | 7 dB | 8°C | 14% | 5 lx | GRASSLAND | INSECTS |
| $o_{12}$ | 8 dB | 9°C | 13% | 0,01 lx | GRASSLAND | INSECTS |
| $o_{13}$ | 4 dB | 42°C | 1% | 85000 lx | DESERT | REPTILES |
| $o_{14}$ | 5 dB | 45°C | 3% | 120000 lx | DESERT | REPTILES |
| $o_{15}$ | 6 dB | 43°C | 2% | 90000 lx | DESERT | REPTILES |
| $o_{16}$ | 3 dB | 46°C | 3% | 4000 lx | DESERT | REPTILES |
| $o_{17}$ | 2 dB | 44°C | 1% | 0,01 lx | DESERT | REPTILES |
| $o_{18}$ | 6 dB | 43°C | 2% | 10 lx | DESERT | INSECTS |

**Fig. 1.** Heterogeneous data running example

## 2 Heterogeneous Data Mining

To the best of our knowledge there is no technique that can detect heterogeneous subspace patterns in high dimensional data. In this section, we discuss existing approaches to mining of novel and interesting patterns in continuous or categorical domains. We analyze their advantages and disadvantages with respect to mining heterogeneous data and propose a consistent model for both data types. We illustrate our discussion with our running example.

### 2.1 Frequent Itemset Mining

For categorical attributes there exist algorithms for mining frequent patterns out of transaction data. A transaction data set consists of sets of items that e.g. in the typical market example are goods that were bought together. A frequent itemset is a combination of several items that were surprisingly often bought together. Each item is given by a categorical value.

Compared to our high dimensional data mining task there are two differences. First, each object in our case has a fixed number of given attributes, while transactions can be of different size. Second, attribute values in the same dimension can not occur together as description of an object, while items can be combined with arbitrary other items in transactions. For both differences we observe that our high dimensional case is a specialization of itemset mining. In addition, both mining tasks have a major point in common. Both have to identify a subset of relevant items (attribute value combinations). For categorical data the objects have exactly the same attribute values in the selected subspace. By counting the frequency of an itemset we thus measure the number of similar objects. This frequency measure is called the support of an itemset. An itemset has then to fulfill a given parameter value of *minSupport* to be frequent.

The frequency measure and its properties are a major topic concerning the efficiency of mining. As a naive approach one would calculate the frequency of all possible item combinations, which is exponential in the number of different items occurring in the data. This is clearly inefficient. A first efficient approach is the Apriori algorithm [9]. It uses a monotonicity property on the frequency measure to efficiently prune the exponential search space. Monotonicity means that removing one item from an itemset $P$, the support of this reduced $P'$ can only be greater or equal to the support of $P$. By removing an item one relaxes the condition for the frequency and thus the number of objects that support this pattern cannot decrease.

**Definition 1 (Monotonicity Property).**

*Given a frequent itemset* $P := \{I_1, \dots I_k\}$*, then all its subsets* $P' \subset P$ *are also frequent itemsets.*

The Apriori algorithm first mines the frequent 1-itemsets and then iteratively combines bottom-up $k$-itemsets to $(k+1)$-itemsets using the monotonicity property to prune non-frequent $(k+1)$-itemsets. More efficient algorithms like the FP-growth algorithm proceed recursively and thus achieve efficient frequent itemset detection without the need of generating all smaller $k$-itemset candidates [10]. Frequent itemset mining is able to mine groups of objects which are identical in a subspace of the high dimensional data. Using a monotonicity property on the frequency measure one can efficiently prune the exponential search space. In addition, indexing the data in the FP-tree provides an efficient detection of frequent itemset without candidate generation. However, the FP-growth algorithm is only applicable to categorical data.

## 2.2 Density-Based Subspace Clustering

For continuous valued attributes it is not meaningful to count the frequencies of attribute values co-occurring in the database. As one has infinitely many possible values per attribute, the probability of finding two data items having the same value is low. Instead a range of values is usually examined. Given the natural order of real values one can specify a neighborhood around an object $O$. Objects inside this neighborhood $N(O)$ can be seen as similar values while objects outside the neighborhood $N(O)$ are dissimilar to the object $O$. This "density-based" paradigm detects arbitrary shaped clusters in continuous data [2].

Further clustering paradigms can be identified in the literature, yet traditional clustering algorithms do not scale to multi-dimensional or high-dimensional spaces. As clusters do not show across all attributes, they are hidden by irrelevant attributes [4]. Dimensionality reduction aims at discarding irrelevant, noisy dimensions [11]. In many practical applications, however, no globally irrelevant dimensions exist.

Consequently, subspace clustering aims at mining clusters in arbitrary, possibly overlapping, subspace projections. As the number of subspaces is exponential in the number of dimensions, this is a challenging task. Therefore, some subspace clustering algorithms discretize the space using grids. As such they can only handle continuous values at the cost of accuracy [5,12]. In contrast, density-based subspace clustering approaches are designed for continuous valued data [6,7,8,13].

Density-based clustering has been shown to successfully identify clusters of arbitrary shape even in noisy settings [2], while discretized and categorical data types conflict with the definition of dense areas in continuous spaces. The density-based paradigm defines clusters as maximal sets of *density-connected* objects. An objects $O$ is *dense* if its neighborhood, specified by an $\varepsilon$-radius around $O$, contains more than the threshold *minPoints* many objects $P$ from database **DB**:

$$|N_\varepsilon(O)| = |\{P \in \mathbf{DB} \mid dist(O,P) \leq \varepsilon\}| \geq minPoints$$

Two dense objects $O$ and $P$ are *density-connected* if there is a chain of dense neighboring objects $Q_1 \dots Q_n$ between them:

$$\exists\ Q_1 \dots Q_n \in \mathbf{DB} : Q_1 = P, Q_n = O, \forall\ i :\ dense(Q_i)$$

$$\wedge\ \forall\ i = 1 \dots n-1 : dist(Q_i, Q_{i+1}) \leq \varepsilon.$$

For subspace clustering, these notions are projected to the respective subspace [14,7,13]. In Figure 2(a) for example, the objects to the right form a 2d cluster $C_4$, as each of them contains many objects within its neighborhood (depicted exemplarily as a circle centered at one of the objects).

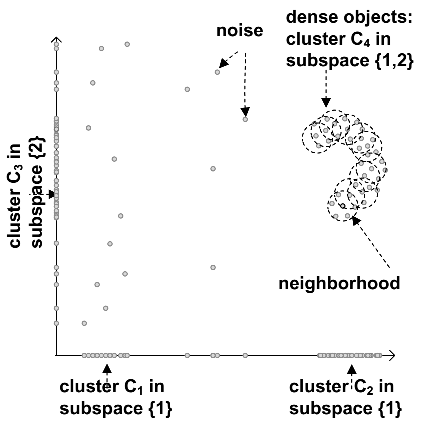
## 2.3 Unbiased Density-Based Subspace Clustering

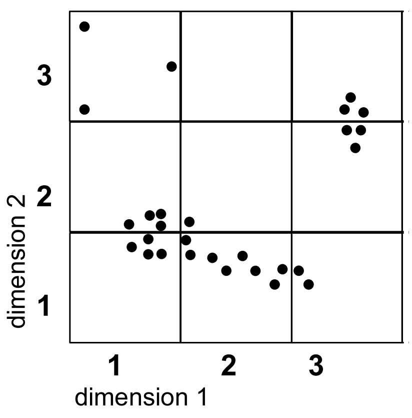
The above definition would result in *dimensionality bias* with respect to different subspace projections: with increasing dimensionality, distances grow and typical densities drop [14,12,7]. Figure 2(a) illustrates this effect: the 2d representation of the data shows a larger spread than the 1d projections at the left and bottom. In general, in high dimensional subspaces, the expected density is smaller than in low dimensional spaces. Consequently, large thresholds are almost never exceeded in high dimensional spaces, resulting in cluster loss. On the other hand, small thresholds that find high dimensional clusters produce tremendous amounts of trivial low dimensional subspace clusters.

In our previous work, we have defined an unbiased density-based subspace clustering for continuous valued data [7]. We therefore normalized the density by the expected density of the subspace dimensionality. The expected density is simply the average number of objects in the $\varepsilon$-neighborhood w.r.t. the dimensionality. This can be computed as the ratio of the volume of the $\varepsilon$-sphere to the volume of the subspace. An object $O$ is *dense* in subspace $\mathbf{S}$ of dimensionality $\mathbf{s}$ if its $\varepsilon$-neighborhood in $\mathbf{S}$ contains more than *minPoints* objects and exceeds the expected density in this subspace by a factor $F$:

$$|N_\varepsilon^{\mathbf{S}}(O)| = |\{P \in \mathbf{DB} \mid dist^{\mathbf{S}}(O,P) \leq \varepsilon\}| \geq$$
$$max\{minPoints, F \cdot expDen(\mathbf{s})\}$$

As subspace clustering algorithms usually generate huge outputs, it is important to exclude *redundant* results. Redundancy means that clusters contain essentially the same information, as they are merely projections of the same pattern in different subspaces. Including redundant subspace clusters in the output is not useful as it leads to overwhelming result sizes without improving the quality of the result [13]. Typically, retaining only maximal subspace clusters, i.e. the highest dimensional projection, leads to good quality of the result. Additionally, this is useful for pruning in depth-first algorithms [13]. An example is included in Figure 2(a): non-redundant result subspace clustering algorithms keep $C_1$ and $C_4$, but not $C_2$ and $C_3$, which are already covered by the maximal $C_4$ in the higher dimensional space.


noise
dense objects: cluster C4 in subspace {1,2}
cluster C3 in subspace {2}
neighborhood
cluster C1 in subspace {1}
cluster C2 in subspace {1}
(a) Density based



(b) Grid based (with discretization)

**Fig. 2.** Subspace clustering

We summarize our unbiased density-based subspace cluster model [7] in the following definition.

**Definition 2.** ***Unbiased Density-Based Subspace Cluster.***
*A set of objects $C \subseteq \mathbf{DB}$ in subspace $\mathbf{S} \subseteq \mathbf{D}$ with $|S| > 1$ and $|C| > minSize$ is a subspace cluster if:*

- *$C$ is **density-connected**: $\forall O, P \in C$: $O^{\mathbf{S}}, P^{\mathbf{S}}$ density-connected w.r.t unbiased density .*
- *$C$ is **maximal**: $\forall O, P \in \mathbf{DB}$: if $O \in C$ and $O^{\mathbf{S}}, P^{\mathbf{S}}$ density-connected then $P \in C$.*
- *$C$ is **non-redundant**: there is no higher-dimensional cluster containing points in $C$.*

In general, the notion of density and density-connectivity is widely used for continuous valued attributes, especially to detect arbitrarily shaped clusters and to achieve a robust behavior in noisy settings. However, due to the density measure it is only applicable to continuous valued attributes. For density computation, database scans are needed which results in high computation costs, especially when compared with the highly efficient frequent itemset mining without candidate generation.

## 2.4 Comparison

Our main aim in this paper is to consistently extend the main properties described in our unbiased subspace cluster definition, for heterogeneous data. For categorical attributes a minimum number of occurrences in the data set makes sense. In frequent itemset mining it corresponds to the *minSupport* a pattern has to fulfill. However, as we mine patterns in arbitrary subspaces, this minimum number has to be normalized according to expected frequency in this subspace. We use a normalized density measure for unbiased density-based subspace clustering to achieve a consistent cluster definition across arbitrary subspaces. For an overall consistent model on heterogeneous data we will extend this model by an unbiased frequency measure in the next section.

Redundancy of subspace clusters can be found in frequent itemset mining as well. It is called closed itemset mining, as one searches for the maximal number of co-occurring items [15,16]. As given by the monotonicity property (cf. Definition 1), all subsets of a frequent itemset are also frequent. The problem in both worlds, frequent itemset mining and subspace clustering, is that one is not interested in such obvious redundant information. The main difference between the two models is the density and frequency notion, as both are related to characteristics of continuous or categorical values. Density connectivity and maximality as given in Definition 2 are not directly applicable for categorical data. To achieve an overall consistent model we have to be aware of these characteristics. Each of them is meaningful in its domain. Thus, our main aim is to ensure these differences without mixing up continuous and categorical attributes by transformation or discretization.

A naive way of integrating both attribute types could be discretization of the continuous attributes, e.g. by grid-based approaches [5,17]. They use discretization either for subspace clustering [5] or for compression and mining of so-called fascicles [17]. However, discretization loses some knowledge about the data, and thus lacks a main property of density-based clustering. Density-based clustering is capable of identifying arbitrarily shaped clusters by taking the neighborhood around objects into account. Grid-based clustering approaches, however, can only detect clusters that consist of cells with more objects than the specified threshold. Consequently, the clustering results of these algorithms are heavily sensitive to the grid resolution and position. Figure 2(b) illustrates this problem: assuming a threshold of at least five objects, the only cluster that would be detected is the one located in cell (2,1). Moreover, this cluster would be revealed only partially, as the remaining parts of the cluster do not exceed the threshold of five objects in their respective cells. Additional post-processing of such clusters could remedy this problem, yet a cluster that is cut apart by the grid such that none of these partitions exceed the threshold (e.g. in cell (3,2) and cell (3,3)) would still be missed completely. If one were to lower the threshold to just three objects, this cluster would be detected, yet more false hits would be produced (e.g. cell (1,3) would produce a pseudo-cluster). The finer grained grid cells are, the less likely distortions are, yet the runtime complexity increases accordingly. Only density-based clustering is capable of detecting all clusters reliably.

## 3 HSM Pattern Model

For categorical data, frequency counting is more meaningful and efficient than the complex density measures, while for continuous data density-based clustering can detect arbitrary shaped clusters and is robust against noise. A combination of both types of models is the goal. We achieve this in an overall consistent manner by incorporating both types of measurements into our HSM pattern model.

In density-based (subspace) clustering, the density of an object $O$ is measured via a density measure $\varphi(O)$. This can be simply the number of objects in the $\varepsilon$-neighborhood of $O$ ($\varphi(O) := |N_\varepsilon^{\mathbf{S}}(O)|$) or any other density measure [7]. Objects in the neighborhood are "density-connected" and assigned to the same cluster. As density has to be comparable i.e. the density measure $\varphi_S$ has to be unbiased with respect to the dimensionality of the subspace $S$ (cf. Definition 2). This is achieved by normalizing with the expected density of the subspace dimensionality [7]:

**Definition 3.** ***Unbiased density normalization.***
*For any density measure $\varphi_S$ with expectation $E[\varphi_S]$,*

$$\frac{\varphi_S}{E\left[\varphi_S\right]} \quad \textit{is dimensionality unbiased.}$$

For continuous attributes, our previous work on dimensionality unbiased subspace clustering provides such an unbiased density measure [7]. Let $E_{cont}\left[\varphi_S\right]$ denote the expected density for a continuous valued subspace $S$. It is computed as the number of objects in the database $DB$ multiplied by the volume ratio of the neighborhood in subspace $S$ to the entire subspace $S$:

**Definition 4.** ***Continuous normalization.***

$$E_{cont}\left[\varphi_S\right] = |DB| \cdot \frac{vol(\varepsilon\text{-}sphere_S)}{vol(S)}$$

For computation details, please refer to [7].

For heterogeneous data, computation of the expected density requires taking categorical attributes into account. By definition, categorical data attributes have no extension, i.e. only discrete values occur. As a consequence, distance values are discrete as well and the notion of $\varepsilon$-sphere neighborhoods leads to discontinuous densities.

We unify density assessment for categorical and continuous attributes. To ensure a consistent density measure, the expected density should be normalized for categorical attributes in the same manner as for continuous attributes. We achieve this consistency by treating categorical values not as discrete points, but as segments of the attribute value range. More precisely, the number of values $v_i$ for each attribute dimension $i$ of the categorical attributes is considered to be the value range extension in this attribute. The overall volume of a categorical subspace $S_{cat}$ is then defined as the product of these ranges, yielding a rectangular overall volume $vol(S_{cat}) = \prod_{i \in S_{cat}} v_i$. The expected density of categorical attributes is then the number of objects in the database $DB$ multiplied by the ratio of the segment volume by the volume of the subspace.

**Definition 5.** ***Categorical normalization.***

$$E_{cat}\left[\varphi_S\right] = |DB| \cdot \frac{vol(segment)}{vol(S)}$$

with $vol(segment) = 1$ as each segment corresponds to one discrete value. For our running example the expected density for the categorical subspace $S_{cat} = \{d_5, d_6\}$ is simple computed by $18 \cdot \frac{1}{3 \cdot 3} = 2$ as we have 18 objects in our database and 3 possible values in each of the two categorical dimensions. This means that we expect to have two objects with the same categorical attribute value combination in our dataset. Thus, a meaningful 2d-categorical cluster in our example should have at least 2 objects, otherwise it has a frequency less than the expected. At least 6 objects form a meaningful 1d-categorical cluster.

This modified density computation corresponds to a frequency count in the categorical attributes, and fits in nicely with our continuous attribute normalization in the sense that the overall expected density normalization $E\left[\varphi_S\right]$ is consistent for both types of attributes in subspace $S = S_{cont} \cup S_{cat}$:

**Definition 6.** ***Heterogeneous normalization.***

$$E\left[\varphi_S\right] = |DB| \cdot \frac{vol(\varepsilon\text{-}sphere_{S_{cont}})}{vol(S_{cont})} \cdot \frac{vol(segment)}{vol(S_{cat})}$$

Thus, the overall normalization measures density of heterogeneous data objects by the degree of deviation from the expected density in continuous attributes and the expected frequency in categorical attributes. We have successfully used this normalization for e.g. ranking of outliers in heterogeneous data [18].

## 4 Efficient Algorithm

Pattern detection in projections of high dimensional data is a computationally challenging task, especially for different data types. Using our consistent density measurement for both continuous and categorical attributes, one has to efficiently mine the heterogeneous subspace patterns. We propose an adaptive algorithm for efficient pattern detection in projections of heterogeneous data. In continuous data, density-based (subspace) clustering is known to be computationally complex, because it involves repeated data scans to identify the objects in the neighborhood [2]. This means that especially for subspace clustering, where the neighborhood has to be computed for different subspace projections, efficiency is a major concern. However, repeated database scans are not required for simple frequency computations on categorical data. Our algorithm for heterogeneous subspace clustering exploits such characteristics of continuous and categorical attributes in a novel index structure. This index is based on our previous work on efficient density-based subspace clustering for continuous values [8] and similar structures for frequent itemset mining [10]. While these techniques are not applicable to heterogeneous data on their own, our index is adaptive to heterogeneous data and yields a significant efficiency improvement compared to recent subspace clustering approaches.

### 4.1 Continuous Attributes

In our previous work, we have developed an indexing structure, the SCY-tree for efficient mining of continuous valued subspace clusters in a depth-first fashion [13,8]. The general idea is to index regions such that they can be evaluated for several different projections, and allow for merging of regions that might contain parts of a subspace cluster.

This indexing structure, however, was devised only for continuous data types. Obviously, we could consider a naive solution of simply treating categorical attributes like continuous ones. However, as we will shortly see in greater detail, this would ignore the very properties of categorical data: merges of adjacent regions cannot occur, as in categorical values, this notion of range within a dimension does not exist. Simply "turning off" the merges would also be far from ideal, as we will see. We will analyze how we can reorganize the index such that heterogeneous data can be mined for subspace clusters in a far more efficient fashion.

But first, we briefly review the SCY-tree. Essentially, the SCY-tree approach benefits from the better efficiency of grid-based algorithms without losing the quality provided by density-based clustering approaches. The idea is to check not only each individual cell against a threshold, but to extend this in a density-based fashion as well: as long as there are objects within a neighborhood distance from the cell's border, we do not discard the cell, but instead merge it with this neighbor. Working in this fashion not only means efficient mining without loss of accuracy, but it also is the basis for our indexing structure that stores the compact cell counts, and also for just-in-time merging and mining of them.

SCY-trees are efficiently constructed by mapping of grid cells and additional neighborhood ranges to index nodes. As discussed before, traditional grids, while showing very good runtimes, lose clusters due to grid resolution and position. This effect can be completely remedied by devising $S$-connectors, i.e. small grid stripes of the size of the neighborhood $\varepsilon$ along the border in each grid cell to its lexicographically larger neighboring cell. Any cluster that would be cut apart in traditional grids would be clearly identified through the presence of an object in the $S$-connector between them. Figure 3 shows the situation in the

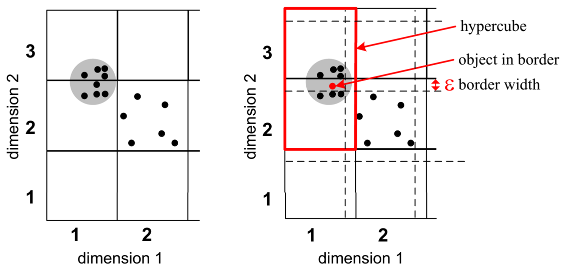


**Fig. 3.** Grid without $S$-connectors and with $S$-connectors

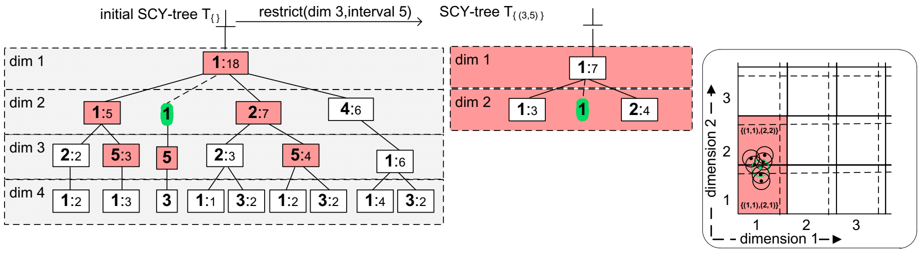


**Fig. 4.** SCY-tree example

traditional grid to the left where the cluster in the grey area would be missed, and the situation with the $S$-connectors where the cluster would be detected after merging of the cells that are connected.

The index therefore not only stores the count of objects in each cell, but additionally information on these $S$-connectors. Very efficient checks of these $S$-connectors allow quick identification of clusters that reach into several cells. These cells are then merged simply by merging of the corresponding node counts in the tree. An example is given in Figure 4 at the right: the two (red) shaded cells in a two-dimensional projection correspond to the SCY-tree in those two dimensions (corresponding to the levels of the tree), and the object highlighted (in green) in the $S$-connector corresponds to a special node in the tree. Each of the nodes represents a cell id in that dimension with the corresponding object count. For example, the cell with id 1 in the first dimension and id 2 in the second dimension contains four objects which are reflected in the leaf entry count of four at the path that starts with cell id 1 in the first dimension and then goes to cell id 2 in the second dimension. Formally, we have:

**Definition 7.** ***SCY-tree structure.***
*A SCY-tree $T_D$ represents a region*
*$D = \{(d_1, i_1), \ldots, (d_k, i_k)\}$ in an arbitrary subspace.*
*The SCY-tree consists of nodes, each of them stores:*

- *a descriptor $(d, i)$ for dimension $d$ and interval $i$ of the region and its count $c$ of objects*
- *a pointer to the parent node and a list of child pointers*
- *a pointer to a linked list of nodes with the same descriptor.*

The basic idea of the algorithm is to identify subspace clusters in a depth-first fashion on the dimensionality of subspaces. By recursively *restricting* the region under study to the relevant dimensions, maximal high dimensional subspace clusters are quickly detected. Going back down to lower dimensional projections, redundant subspace clusters can be immediately pruned. Restriction on SCY-trees is easy, as it corresponds to simply reading off the relevant paths in the tree. The key idea is to perform all the mining only on this tree structure and thus avoid expensive database scans. After having build the initial SCY-tree with only one database scan all the needed information is covered by the SCY-tree. We explain this procedure with our running example. On the left side of Figure 4 we see the initial SCY-tree, which represents the four-dimensional continuous space of our example. To compute the number of objects in any cell, for example cell 5 in dimension three, simply read the values that correspond to any path with id 5 in the third dimension. The distribution in dimensions one and two is then represented as the restricted SCY-tree to the right. This restriction can be performed recursively. Note that the order of dimensions is not important for the accuracy of the result, even dimensions in-between can be simply disregarded while all relevant paths for other projections are extracted.

## 4.2 Categorical Attributes

As mentioned in Section 2.1 for categorical data the FP-tree mines frequent itemsets efficiently without candidate generation [10]. The basic idea is to have all the needed information in this index structure to avoid expensive database scans. As in the SCY-tree after building the initial tree, mining is performed only on the tree structure. In Figure 5 we see the initial FP-tree for our running example. As the FP-tree can only handle categorical data we hide the first four continuous dimensions as one item "continuous". Proceeding recursively, the FP-growth algorithm restricts the initial FP-tree e.g. as depicted in the item "REPTILES" by extracting all paths ending in a node labeled with that item.

Both the SCY-tree and the FP-tree from frequent itemset mining are related in the sense that they operate on compact node entries that represent cell counts. The SCY-tree, however, is capable of dealing with continuous values as well, though merging of neighboring cells where necessary. However, such merge operations lead to computational overhead. Mining continuous data on the SCY-tree has higher runtime than mining on the FP-tree. The FP-tree, for both attribute types, on the other hand, is not able to handle continuous values.

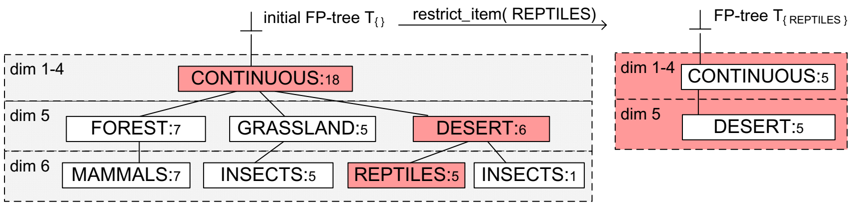


**Fig. 5.** FP-tree example

## 5 Heterogeneous Attributes (HSM-Tree)

As a straightforward solution to mining of heterogeneous data, we simply extend the ideas of SCY-tree [8] and FP-tree [10] (cf. reviews in Sec. 4.1 and 4.2) to heterogeneous data. Our novel HSM-tree structure can handle both categorical and continuous data types by automatically adapting to the actual data type. This adaption makes the HSM-tree a non-trivial combination of two known tree structures. Furthermore, we introduce two novel strategies for adapting to different data types during tree construction.

**Definition 8.** ***HSM-tree structure.***
*A HSM-tree* $T_{HD}$ *represents a heterogeneous region* $HD = \{hd_1, \ldots, hd_k\}$*.*
*Each level in the HSM-tree represents one dimension. The order of dimensions* $st(d_1) \ldots st(d_k)$ *is given by (a subset of) the order* $st(d_i) \leq st(d_j)\ \forall i < j$ *with respect to strategy* $st(d_i)$*.*
*The HSM-tree consists of nodes, each of them stores:*

– *a descriptor* $(d, i)$ *for dimension* $d$ *and*
  - *if* $d$ *is continuous, interval* $i$ *of the region and its count* $c$ *of objects*
  - *if* $d$ *is categorical, item* $i$ *and its frequency* $c$
– *a pointer to the parent node and a list of child pointers*
– *a pointer to a linked list of nodes with the same descriptor.*

As one can easily see by comparing the above definition of the HSM-tree with the SCY-tree definition (Def. 7), we have two major differences: First, when building the HSM-tree, we differentiate between continuous and categorical attributes to compute the count for a region, or to record the frequency, respectively. Second, the HSM-tree follows a strategy (cf. Sec. 5.2) for building the tree. This strategy defines a favorable ordering of the dimensions for efficient mining. Please note that two major changes of HSM-tree are not explicit in the above definition: During mining on the HSM-tree, we use the consistent density normalization for heterogeneous data (cf. Def. 6), and, most importantly, we adapt to the heterogeneous attributes indicated by the descriptors in the HSM-tree (cf. following sections).

### 5.1 Adaptive Mining

Based on the simple HSM-tree structure we perform a novel adaptive mining. Adaption is required as there is a fundamental difference between the data types that merits more detailed consideration: There is no notion of ranges in categorical values. That is, only in continuous values a cluster might stretch across several grid cells in one dimension, because the values of the objects are neighboring, e.g. show temperature values that do not differ much. In categorical dimensions, this does not occur, e.g. either we are dealing with an insect or with a mammal, but in general there is no notion about "closeness" or distance between the two categorical values. This results in two effects for categorical values in HSM-trees: first, we do not need to check $S$-connectors, and second, we do not need to merge "cells" (i.e. attribute value combinations).

However, just by eliminating the corresponding nodes for $S$-connectors and omitting checks for merges, we are not doing as good as possible. We propose to reorganize the tree to achieve far better runtimes of the mining process on the HSM-tree. Basically, we observe that, the order of the dimensions in the HSM-tree matters with respect to runtime performance. Mining starts with the leaf nodes to extract the longest paths and thus also the high dimensional patterns. Thus having categorical values which are highly clustered at the leaf level would be beneficial for efficient mining.

Looking again at our running example we can construct the restricted HSM-tree for "MAMMALS" as shown in Figure 6. It also illustrates the advantage of combining both continuous and categorical information in one tree structure. Having restricted the initial HSM-tree already to one categorical value, the highly correlated attributes vegetation and prevailing animals directly form a cluster. We only see "FOREST" nodes at the new leaf level, and all objects being represented by this HSM-tree also contribute to the hidden cluster. Furthermore, after the first restrictions in categorical values we obtain small tree structures. This is due to the fact that no merge operations have to be performed on these trees.

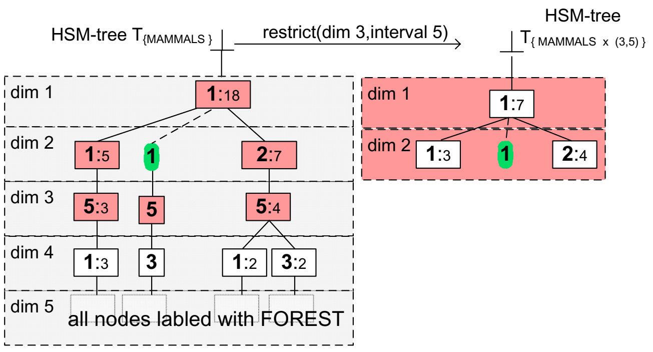


**Fig. 6.** HSM-tree example

Adaptation to restrictions in both categorical and continuous attributes is the key characteristic of adaptive mining using the HSM-tree. Thus, our novel tree structure enables the mining in heterogeneous subspaces of the data. Thus, in our running example, we can mine the heterogeneous subspace "animals" and "humidity" by restriction operations on the HSM-tree (cf. Figure 6). Performing another restriction on a different attribute type (e.g. in cell 5 in the continuous dimension 3) is quite easy as the whole mining process relies on a common basis. We thus easily can mine a heterogeneous pattern e.g. "MAMMALS" x (3,5) which represents the sensors surrounded by mammals and having a high humidity around 50%. The frequency/density of this attribute value combination can easily be determined based on the given HSM-tree. The sum of all nodels labeled with 5 in dimension 3 is equal to 7 (i.e. 7 objects support this pattern).

## 5.2 Adaptive Tree Construction

In construction of HSM trees, our goal is to ensure a structure that allows efficient mining of arbitrary patterns in heterogeneous attributes. According to Definition 8 the dimensions are ordered $d_1 \dots d_k$ with descriptors of $d_1$ at the root node and descriptors of $d_k$ at the leafs of the HSM-tree. The order is defined by a sorting criterion $st(d_i)$ such that the ordered dimensions fulfill $st(d_i) \leq st(d_j)\ \forall i < j$. We study two different strategies for efficient mining. Both defining a different sorting of the dimensions in the HSM-tree.

**Strategy 1: Reducing the tree size**
To reduce the size of the tree, one would sort dimensions with respect to the scattering of data in this dimension. A dimension in which the data is distributed over a wide range should not be inserted first in the tree as it would create a branching at the top of the tree. As an extreme, we would not want to first insert all continuous dimensions and then all categorical dimensions. This would clearly constitute a worst case scenario with respect to the resulting tree size. Instead, a dimension in which the data is clustered in one region leads to many common paths in the tree when inserted first. Many common paths reduce the size of the HSM-tree. In our first strategy, the idea is therefore to start by inserting the categorical dimensions to minimize tree size.

**Strategy 2: Reducing the number of merges**
As a drawback to the previous strategy, however, we observe that mining scattered dimensions with noisy data first forces many merges on large HSM-trees as no other dimensions have been restricted yet. These merges might not even lead to subspace clusters in the end as the *minSize* threshold might only be exceeded due to the large size of the tree. Many merges can be avoided through a different processing order. By mining the scattered continuous dimensions last, we need to perform merges only on trees that have been restricted in several dimensions and are thus small. Merges on small HSM-trees are faster which is beneficial for the overall runtime. In this strategy, we focus on avoiding merges, and thus we insert the dimensions that contain clustered data or contain only categorical values last, i.e. in the opposite order as in the first strategy.

To detect scatter, we use entropy as a measure for both strategies. Entropy is an information theoretic indicator for the homogeneity of the data and can be used to assess the distribution of objects in each dimension. Given the discretized data space for each dimension and thus also the percentage $f_i$ of objects in each interval $i = 1 \dots k$ entropy is calculated by $E(f_1, \dots, f_k) = -\sum_{i=1}^{k} f_i \cdot \log(f_i)$. Inserting dimensions with low entropy values first (ascending order) realizes the first sorting strategy for small trees, while inserting dimensions with high entropy values (descending order) realizes the second strategy for fast subspace mining. Both ascending and descending order will be analyzed in experiments in the following section.

An overview over mining on HSM-trees is given in Algorithm 1. For any database, the algorithm uses only two database scans to build the HSM-tree. One scan for the computation of the dimension order with respect to the chosen strategy and a second scan to create the initial HSM-tree. To reduce the number of merge operations in later mining steps, we clearly differentiate between continuous and categorical attributes. In the mining phase, we proceed recursively on the dimensionality for any region that might contain subspace clusters. For any subspace region, we determine its unbiased density value according to the consistent definition for heterogeneous data given in Def. 6, and merge neighboring regions only where necessary (i.e. in the continuous attributes). The result of our algorithm is the set of all non-redundant heterogeneous subspace clusters, detected efficiently using the HSM-tree.

**Algorithm 1.** Heterogeneous Subspace Mining on HSM-trees

```
    Algorithm HSM ( Database $DB$, Strategy $st$, set of dimensions $D$ )
 1: compute order $d_1, \ldots, d_k$, using strategy criterion $st(d_i)$      ▷ first scan of $DB$
 2: for each object in $DB$ do                                                ▷ second scan of $DB$
 3:     for $i$=1 to $k$ do
 4:         if $d_i$ continuous then
 5:             insert grid cell and object count into initial HSM-tree
 6:         else if $d_i$ categorical
 7:             insert item and frequency into initial HSM-tree
                                                                          ▷ end of building phase
 8: start recursive mining with initial HSM-tree $T_{\{\}}$
    in each recursive step with given HSM-tree $T_{HD}$ do:
 9: for each descriptor $(d, i) \in HD$ in the HSM-tree $T_{HD}$ do
10:     restrict to $T_{HD \cup (d,i)}$
11:     compute density in merged grid cells of $T_{HD \cup (d,i)}$
12:     compute frequency for categorical items in $T_{HD \cup (d,i)}$
13:     in each subspace, use heterogeneous normalization as in Def. 6
14:     recursive step for $T_{HD \cup (d,i)}$ until no further restriction possible
                                                                          ▷ end of mining phase
15: output all detected heterogeneous subspace clusters
```

## 6 Experiments

We first show scalability of our heterogeneous subspace mining approach. As no other subspace mining algorithms exist on heterogeneous high dimensional data, we compare with two recent subspace clustering algorithms on continuous valued attributes. In a more detailed analysis we evaluate the impact of the heterogeneity of the data on the runtime of HSM. We show that our heterogeneous approach achieves efficiency improvement on categorical data by orders of magnitude. Due to the adaptive index structure we achieve improvements for the categorical parts of the data. On continuous attributes we can further improve efficiency by strategies for tree construction. For a thorough analysis, we evaluate runtime and memory performance of HSM with both ascending and descending sorting strategies.

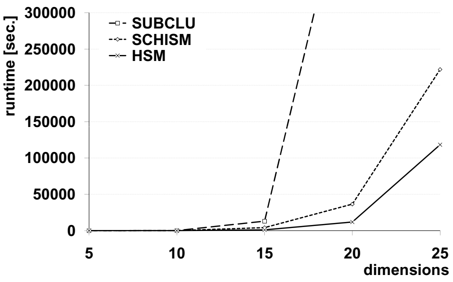
## Synthetic data setup

We generate synthetic data for scalability experiments following a method proposed in [6,7] to generate density-based clusters in arbitrary subspaces. Given the subspace and the number of objects for each cluster, the generator creates dense regions separated by noisy regions. In addition, our generator takes into account that objects can belong to multiple subspace clusters, just as in most real world data sets. We generate data of different dimensionalities and hide subspace clusters with a maximal dimensionality of 80% of the data dimensionality. Subspace clusters are hidden in the synthetic data with partial overlapping of objects.

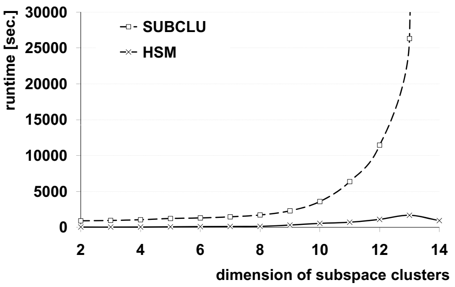
## Scalability w.r.t dimensionality

In our first experiment, we evaluate scalability with respect to the dimensionality of the subspace. As subspace clustering aims to detect patterns in high-dimensional data scalability is crucial. Figure 7(a) illustrates all three algorithms, HSM, SUBCLU and SCHISM, on datasets of dimensionalities 5 to 25. As we can see, SUBCLU, an apriori-based algorithm, deteriorates extremely at more than about 15 dimensions. SCHISM, a grid-based subspace clustering algorithm, performs much better than SUBCLU. Our HSM approach, however, clearly outperforms both competitors. HSM thus shows far better runtime performance than SCHISM, even though SCHISM is an approximative approach that does not mine the full result set.

Figure 7(b) illustrates the effect of varying subspace cluster dimensionalities. In a database of fixed dimensionality 15, subspace cluster dimensionality is varied from 2 to 15. As we can see, dimensionality of the subspace clusters is the decisive factor in runtime performance of SUBCLU. It shows reasonable runtimes up to about 10-dimensional subspace clusters. The breadth-first SUBCLU algorithm generates all lower dimensional projections, thus does not scale beyond this point. HSM is only slightly affected by the dimensionality of subspace clusters as it avoids unnecessary generation of lower dimensional projections by its compact tree structure.



(a) Data space



(b) Hidden clusters

**Fig. 7.** Scalability w.r.t dimensionality

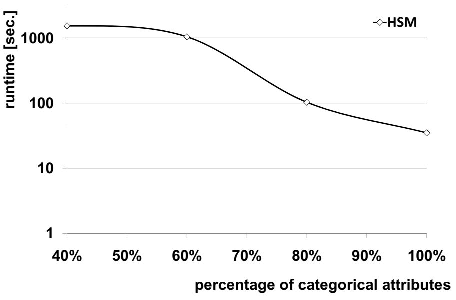


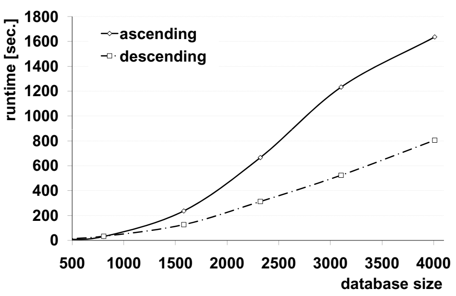
**Fig. 8.** Variation of attribute types

## Scalability w.r.t heterogeneity

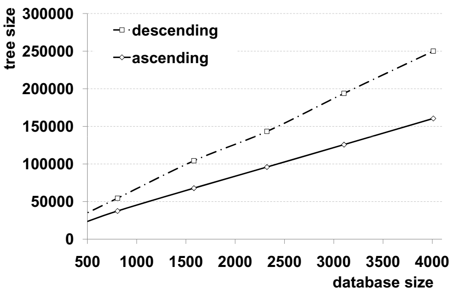
We increase the percentage of categorical data to evaluate how the runtime of our HSM approach is affected by more and more categorical attributes. Due to the simpler frequency measurement in categorical data the algorithm should perform even better. In Figurer 8 we observe an efficiency improvement by orders of magnitude as we increase the percentage of categorical attributes. Obviously the HSM algorithm is able to adapt to the heterogeneous data. Being aware of the fact that for categorical data frequency is meaningful but also very efficient HSM achieves a significant runtime improvement for heterogeneous data.

## Sorting strategies

We next evaluate the effect of different strategies for constructing the tree structure. As mentioned, the index construction mainly depends on the order of the dimensions. Two strategies have been proposed in the last section: creating compact trees using an ascending order of the dimensions w.r.t. their entropy or avoiding early merges by sorting the dimensions in descending order w.r.t. their entropy. We first evaluate the effect of the different sorting strategies on the runtime (see Figure 9(a)). Clearly, avoiding early merges improves the overall runtime of HSM. Sorting in descending order almost halves the runtime.



(a) Runtimes



(b) Tree size

**Fig. 9.** Sorting strategies

On the other hand, by sorting in ascending order the size of the initial tree is reduced as presented in Figure 9(b). This shows that even with a smaller initial tree the mining algorithm cannot reach the good runtimes of the descending order heuristic. For efficient mining it is thus essential to have as few merges as possible in the first restrictions. This is achieved by the descending order heuristic.

## 7 Conclusion

In this work, we have discussed data mining in heterogeneous data. We have presented a model that bridges the gap between continuous and categorical data types by providing a consistent view on interesting groups ob data objects. Our approach takes the expected density and the expected frequency of the subspace projections into account to uncover truly interesting patterns. Our algorithmic approach makes best use of indexing possibilities by identifying suitable ordering of the dimensions to achieve low runtimes. Our experiments demonstrate that our technique for heterogeneous data mining outperforms existing techniques. By adapting to the different characteristics of attribute types we further achieve a significant efficiency improvement for heterogeneous data.

## Acknowledgments

This research was funded in part by the cluster of excellence on Ultra-high speed Mobile Information and Communication (UMIC) of the DFG (German Research Foundation grant EXC 89).

## References

1. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco (2001)
2. Ester, M., Kriegel, H.-P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases. In: KDD, pp. 226–231 (1996)
3. Zaki, M., Peters, M., Assent, I., Seidl, T.: CLICKS: An effective algorithm for mining subspace clusters in categorical datasets. DKE 60, 51–70 (2007)
4. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is nearest neighbors meaningful. In: IDBT, pp. 217–235 (1999)
5. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications. In: SIGMOD, pp. 94–105 (1998)
6. Kailing, K., Kriegel, H.-P., Kröger, P.: Density-connected subspace clustering for high-dimensional data. In: SDM, pp. 246–257 (2004)
7. Assent, I., Krieger, R., Müller, E., Seidl, T.: DUSC: Dimensionality unbiased subspace clustering. In: ICDM, pp. 409–414 (2007)
8. Assent, I., Krieger, R., Müller, E., Seidl, T.: INSCY: Indexing subspace clusters with in-process-removal of redundancy. In: ICDM, pp. 719–724 (2008)

9. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: VLDB, pp. 487–499 (1994)
10. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: SIGMOD, pp. 1–12 (2000)
11. Joliffe, I.: Principal Component Analysis. Springer, New York (1986)
12. Sequeira, K., Zaki, M.: SCHISM: A new approach for interesting subspace mining. In: ICDM, pp. 186–193 (2004)
13. Assent, I., Krieger, R., Müller, E., Seidl, T.: EDSC: Efficient density-based subspace clustering. In: CIKM, pp. 1093–1102 (2008)
14. Kailing, K., Kriegel, H.-P., Kröger, P., Wanka, S.: Ranking interesting subspaces for clustering high dimensional data. In: Lavrač, N., Gamberger, D., Todorovski, L., Blockeel, H. (eds.) PKDD 2003. LNCS, vol. 2838, pp. 241–252. Springer, Heidelberg (2003)
15. Zaki, M.J.: Generating non-redundant association rules. In: SIGKDD, pp. 34–43 (2000)
16. Pei, J., Han, J., Mao, R.: CLOSET: An efficient algorithm for mining frequent closed itemsets. In: SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 21–30 (2000)
17. Jagadish, H.V., Madar, J., Ng, R.T.: Semantic compression and pattern extraction with fascicles. In: VLDB, pp. 186–198 (1999)
18. Müller, E., Assent, I., Steinhausen, U., Seidl, T.: OutRank: ranking outliers in high dimensional data. In: DBRank at ICDE, pp. 600–603 (2008)

# Split-Order Distance for Clustering and Classification Hierarchies

Qi Zhang, Eric Yi Liu, Abhishek Sarkar, and Wei Wang

Department of Computer Science
University of North Carolina at Chapel Hill
{zhangq,liuyi,asarkar,weiwang}@cs.unc.edu

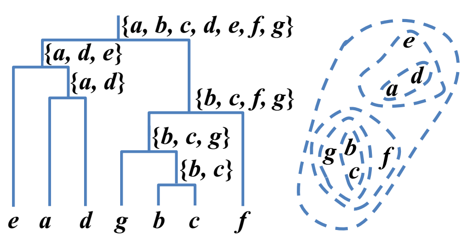
**Abstract.** Clustering and classification hierarchies are organizational structures of a set of objects. Multiple hierarchies may be derived over the same set of objects, which makes distance computation between hierarchies an important task. In this paper, we model the classification and clustering hierarchies as rooted, leaf-labeled, unordered trees. We propose a novel distance metric Split-Order distance to evaluate the organizational structure difference between two hierarchies over the same set of leaf objects. Split-Order distance reflects the order in which subsets of the tree leaves are differentiated from each other and can be used to explain the relationships between the leaf objects. We also propose an efficient algorithm for computing Split-Order distance between two trees in $O(n^2d^4)$ time, where $n$ is the number of leaves, and $d$ is the maximum number of children of any node. Our experiments on both real and synthetic data demonstrate the efficiency and effectiveness of our algorithm.

**Keywords:** Clustering Hierarchy, Classification Hierarchy, Tree Distance.

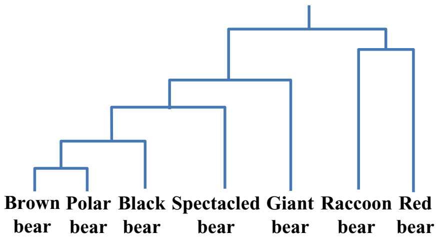
## 1 Introduction

Clustering and classification hierarchies are important tools for capturing the relationships among objects. Two representative examples are *dendrograms* illustrating the hierarchical clustering result (Fig. 1(a)) and *taxonomies* classifying the biological species (Fig. 1(b)). A hierarchy organizes the set of objects in a tree, where objects with higher similarity are grouped together at a lower level, before more distant objects merge into bigger groups. The groups at the intermediate levels represent subclusters or subclasses.

Clustering and classification hierarchies can be automatically constructed given the pair-wise distance matrix over the set of objects. However, different models and methods may yield different hierarchical structures. Quantifying the differences between hierarchies becomes crucial for tasks such as summarization of hierarchical patterns, computation of consensus hierarchies, and comparison of hierarchically structured data. A motivating example is the comparison of *phylogenetic taxomonies*, or *phylogeny trees*. A phylogeny tree (Fig. 1(b)) describes the evolutionary relationship between different organisms. The leaves of



(a) *A dendrogram over 7 objects*



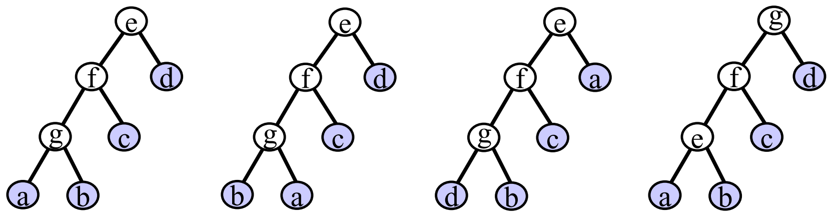
(b) *A taxnomony over 7 species*

**Fig. 1.** Example clustering and classification hierarchies

the tree represent the species, and the internal nodes correspond to the specialization events, where the evolution diverges in different directions to generate subspecies. The root of the tree is the most recent common ancestor for all the species. Different hierarchies can be derived for a given set of species using different phylogeny tree construction algorithms, such as UPGMA [16], neighbor-join [14], maximum parsimony [6], etc. Comparing the similarity of different phylogeny trees is important for evaluating different algorithms and deriving the consensus phylogeny tree.

Many algorithms have been proposed for comparing general tree topologies. Tree edit distance is a classic metric to compare two trees with both internal nodes and leaf nodes labeled. Different from a general tree structure, clustering and classification hierarchies are leaf-labeled trees; leaves represent the objects. The tree edit distance computation for unordered trees is NP-complete [21]. Polynomial-time algorithms [21,3,10,11,18] only exist for ordered trees. As for classification or clustering hierarchies, there is no specific order among siblings. If we were to consider classification or clustering hierarchies as ordered, fully-labeled trees, metrics such as the tree edit distance may produce counter-intuitive results. Consider four trees presented in Fig. 2. $T_1$, $T_2$, $T_3$, and $T_4$ are different hierarchies over a common object set $\{a, b, c, d\}$. To apply the ordered tree edit distance, we also impose labels for the internal nodes. As classification hierarchies, $T_2$ and $T_4$ are the same as $T_1$. The objects are classified in the same way in each tree; specifically, the order in which the objects are differentiated from each other is the same. However, the tree edit distance between $T_1$ and $T_2$ is 2 since leaves $a$ and $b$ are transposed. The tree edit distance between $T_1$ and $T_4$ is also 2 since two internal nodes are transposed. In addition, $T_3$ has a tree edit distance of 2 from $T_1$, which implies that $T_1$, $T_2$, and $T_3$ are equidistant from $T_1$. However, the way in which $a$ and $d$ are classified in $T_3$ is very different from how they are classified in $T_1$, even though $a$ and $d$ are classified in the same way in both $T_1$ and $T_2$.

In this paper, we propose a novel distance metric, Split-Order distance, for comparing clustering or classification hierarchies. Different from a general tree structure, a clustering or classification hierarchy can be modeled as rooted, unordered, leaf-labeled trees. Essential to these hierarchical structures is the set of relationships among the leaf objects captured by the hierarchy: an object is more closely related to another object which it merges with at a lower level than



(a) $T_1$ (b) $T_2$ (c) $T_3$ (d) $T_4$

**Fig. 2.** An example with four trees. $T_1$, $T_2$, $T_3$, and $T_4$ are four hierarchies with the same set of objects(leaf labels) $\{a, b, c, d\}$. To apply the tree edit distance, we also impose labels for internal nodes. Trees are considered as ordered.

another object which it merges with at a higher level. We refer to a *split* between two objects as the smallest subcluster or subclass in the hierarchy where both objects belong to. A split corresponds to an internal node in the tree which is the most recent common ancestor of the two leaf nodes representing the two objects. All the splits form a partial order which uniquely determines the relationship among all the objects captured by the hierarchy. For example, in Fig. 1(a), the split between $b$ and $c$ happens at a lower level than the split between $c$ and $f$ does; in Fig. 1(b), the split between polar bear and red bear occurs at a higher level than the split between polar bear and brown bear. We define the Split-Order distance between two hierarchical structures based on the order of the splits occuring in the tree. Our contributions can be summarized as follows:

1. We define a novel distance metric, Split-Order distance, between two clustering or classification hierarchies. We prove that Split-Order distance is a metric.
2. We prove that a complete set of split orders uniquely defines a hierarchical structure. In addition, we propose an algorithm for reconstructing the hierarchy using a set of split orders.
3. We propose an efficient algorithm for computing the Split-Order distance between any two hierarchies over the same set of objects. Our algorithm takes $O(n^2d^4)$ time, where $n$ is the number of leaves, $d$ is the maximum degree of any node.

The rest of the paper is organized as follows. We review the related work in Section 2 and introduce the preliminaries in Section 3. In Section 4, we discuss the formal definition of Split-Order distance. We present our algorithm for efficient computation of the Split-Order distance in Section 5. The experimental results are reported in Section 6, and Section 7 concludes the paper.

## 2 Related Work

Many algorithms have been proposed for comparing tree-like or hierarchically structured data. One of the tree distance metrics which has been extensively studied is the tree edit distance. The tree edit distance evaluates the cost of transforming a tree into another tree through a sequence of operations, such as deleting, inserting, and relabeling nodes. A cost is defined for each operation, and the minimum of the total cost of all operations in a sequence is the tree edit distance. Among the different tree edit distance computation algorithms [4,17,3,10,11,18,21], two representative ones are Zhang-Shasha [21] and Klein [10]. Both algorithms use dynamic programming techniques with worst-case complexities $O(n^4)$ and $O(n^3 logn)$, respectively. Besides tree edit distance, another category of tree distance metrics compare the trees based on the structures they share such as maximum agreement subtrees [2], cousin-pairs [15], etc. However, as mentioned in the previous section, these general tree distance metrics may not be readily used for comparing clustering or classification hierarchies which are usually modeled as rooted, unordered, leaf-labeled trees, and are able to capture the relationship among leaf objects. For phylogeny trees, several distance metrics have been proposed incorporating biologically meaningful definitions [19,13,12,1,5]. But they may not work well on general clustering and classification hierarchies.

## 3 Preliminaries

We model the clustering and classification hierarchies as rooted, unordered, leaf-labeled trees, as shown in Fig. 3.

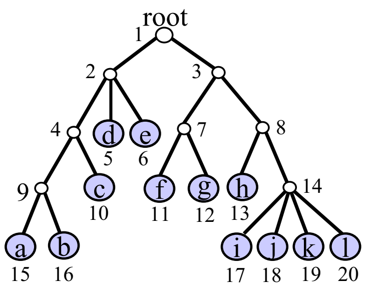


**Fig. 3.** A rooted, unordered, leaf-labeled tree representing a clustering or classification hierarchy over a set of objects $\Sigma = \{a, b, c, d, e, f, g, h, i, j, k, l\}$

The leaf nodes represent the objects. Each internal node represents the split of a cluster or class into several subclusters or subclasses. The root represents the entire set of objects. We consider unordered trees, since the sibling order does not exist in classification or clustering hierarchies. We also assume that each internal node has at least two children; each child represents a different subclass or subcluster resulting from the split.

Let the set of leaf labels be $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$. We refer to a leaf node using its label in the following discussion. All nodes in the tree are uniquely numbered for easy reference (see Fig. 3). For a leaf node $\sigma_i \in \Sigma$, the internal nodes on the path from $\sigma_i$ to the root are ancestors of $\sigma_i$. The ancestors represent the subclusters or subclasses containing $\sigma_i$. For any two leaf nodes $\sigma_i$, $\sigma_j$, we define the *Split* between $\sigma_i$, $\sigma_j$ as follows:

**Definition 1.** *The* ***Split*** *between $\sigma_i$, $\sigma_j$ (denoted as $Split(\sigma_i, \sigma_j)$). Given a rooted, unordered tree $T$ with a set of leaf labels $\Sigma$, for any two leaf nodes $\sigma_i, \sigma_j \in \Sigma$, the Split between $\sigma_i$, $\sigma_j$ is defined as the most recent common ancestor of $\sigma_i$ and $\sigma_j$ in $T$.*

$Split(\sigma_i, \sigma_j)$ represents the smallest cluster or class which includes both $\sigma_i$ and $\sigma_j$, *i.e.*, the split during the top-down hierarchical clustering or classification process which divides $\sigma_i$, $\sigma_j$ into different subclusters or classes. For example, in Fig. 3, nodes 1, 2, 4, and 9 are ancestors of leaf node $a$, nodes 1, 2, and 4 are ancestors of leaf node $c$. Therefore, $Split(a, c)$ is node 4.

In the following section, we define the Split-Order distance for any two hierarchies $T_1$, $T_2$ over the same set of leaf objects $\Sigma$.

## 4 Split-Order Distance

Given a rooted, unordered tree $T$ with a set of leaf labels $\Sigma$, for any leaf node $\sigma_i$, the order of the splits which separate $\sigma_i$ and the remaining leaf nodes determines the relationship between $\sigma_i$ and other leaves. By "order", we mean that the split happens "earlier" (closer to the root) or "later" (closer to the leaf $\sigma_i$). Formally, for any leaf node $\sigma_i$, we define the order relationship of any two splits $Split(\sigma_i, \sigma_j)$,$Split(\sigma_i, \sigma_k)$ as follows:

**Definition 2.** *Split-Order Relations. Given a rooted, unordered tree with a set of leaf labels $\Sigma$, for any leaf node $\sigma_i \in \Sigma$, and any two other leaf nodes $\sigma_j$, $\sigma_k \in \Sigma$, we define*

1. *$SplitOrder(\sigma_i, \sigma_j, \sigma_k)$ = '$\prec$', if $Split(\sigma_i, \sigma_j)$ is an ancestor of $Split(\sigma_i, \sigma_k)$, ($Split(\sigma_i, \sigma_j)$ happens earlier than $Split(\sigma_i, \sigma_k)$);*
2. *$SplitOrder(\sigma_i, \sigma_j, \sigma_k)$ = '$\succ$', if $Split(\sigma_i, \sigma_k)$ is an ancestor of $Split(\sigma_i, \sigma_j)$, ($Split(\sigma_i, \sigma_j)$ happens later than $Split(\sigma_i, \sigma_k)$);*
3. *$SplitOrder(\sigma_i, \sigma_j, \sigma_k)$ = '$=$', if $Split(\sigma_i, \sigma_j) = Split(\sigma_i, \sigma_k)$, ($Split(\sigma_i, \sigma_j)$ happens at the same time with $Split(\sigma_i, \sigma_k)$).*

For example, in Fig. 3, we have $SplitOrder(a, b, c)$ = '$\succ$', $SplitOrder(a, g, d)$ = '$\prec$', and $SplitOrder(a, f, h)$ = '$=$'. The topology of $T$ determines a map $\Sigma \times \Sigma \times \Sigma \mapsto \{\prec, \succ, =\}$. In the following discussion, we show that the complete set of the Split-Order relations $\Theta = \{SplitOrder(\sigma_i, \sigma_j, \sigma_k) = \theta_{i,j,k}\}, \forall \sigma_i, \sigma_j, \sigma_k \in \Sigma$, $\theta_{i,j,k} \in \{\prec, \succ, =\}$ can serve as a unique signature of $T$.

**Theorem 1.** *Given a complete set of the Split-Order relations $\Theta = \{SplitOrder(\sigma_i, \sigma_j, \sigma_k) = \theta_{i,j,k}\}, \forall \sigma_i, \sigma_j, \sigma_k \in \Sigma$, $\theta_{i,j,k} \in \{\prec, \succ, =\}$, we can either reconstruct a unique tree or declare that a tree corresponding to $\Theta$ does not exist.*

*Proof.* We prove the theorem by describing the following recursive algorithm to reconstruct a unique tree or report the nonexistence of a tree.

In the beginning, we start with the set of leaf nodes $\Sigma$. All the nodes are unprocessed initially. Each time we pick an unprocessed leave node from $\Sigma$ for processing until all nodes in $\Sigma$ are processed. Let $\sigma_i$ be the node we pick. We find the set of leaf nodes $\{\sigma_j\}$ which have the latest split with $\sigma_i$:

$$\{\sigma_j | \forall \sigma_k \in \Sigma, \sigma_k \neq \sigma_i, \sigma_k \neq \sigma_j, SplitOrder(\sigma_i, \sigma_j, \sigma_k) = '\prec'\} \tag{1}$$

It is intuitive to prove that $\sigma_i$ and all nodes in $\{\sigma_j\}$ must be siblings. Next, we examine whether there are any conflicting Split-Order relations in $\Theta$. For any two leaf nodes $\sigma_{j_1}, \sigma_{j_2} \in \{\sigma_j\}$, we check whether the following two equations hold:

$$\begin{aligned} &SplitOrder(\sigma_i, \sigma_{j_1}, \sigma_k) = SplitOrder(\sigma_i, \sigma_{j_2}, \sigma_k) \\ &\forall \sigma_k \in \Sigma \end{aligned} \tag{2}$$

$$\begin{aligned} &SplitOrder(\sigma_{j_1}, \sigma_k, \sigma_l) = SplitOrder(\sigma_{j_2}, \sigma_k, \sigma_l) = SplitOrder(\sigma_i, \sigma_k, \sigma_l) \\ &\forall \sigma_k, \sigma_l \in \Sigma \end{aligned} \tag{3}$$

If any of the above equations does not hold, which implies conflicting Split-Order relations in $\Sigma$, a tree conforming to $\Sigma$ does not exist. Due to limited space, we omit the proof here. If both equations hold, we create a parent node $\sigma_{new}$ for $\sigma_i$ and all leaf nodes in $\{\sigma_j\}$. We add $\sigma_{new}$ to $\Sigma^{(1)}$, and mark $\{\sigma_j\}$ and $\sigma_i$ as processed.

If $\Sigma$ still contains unprocessed nodes, we pick another leaf node, and start the process again. After all nodes in $\Sigma$ have been processed, we start the next iteration with $\Sigma^{(1)}$. All nodes in $\Sigma^{(1)}$ will be treated as leaf nodes, and their Split-Order relations are determined by the nodes they represent, as follows:

$$SplitOrder(\sigma_{u'}^{(1)}, \sigma_{v'}^{(1)}, \sigma_{w'}^{(1)}) = SplitOrder(\sigma_u, \sigma_v, \sigma_w) \tag{4}$$

where $\sigma_u$ is any child of $\sigma_{u'}^{(1)}$, $\sigma_v$ is any child of $\sigma_{v'}^{(1)}$, and $\sigma_w$ is any child of $\sigma_{w'}^{(1)}$.

We recurse until at iteration $p$, $\Sigma^{(p)}$ contains only the root node. If at any point, either Equation 2 or 3 does not hold, the process aborts and reports the nonexistence of a tree. The complete algorithm is shown in Algorithm 1.

**Definition 3.** *Split-Order distance. For two rooted, unordered trees $T$, $T'$ with the same set of leaf labels $\Sigma$, and their corresponding Split-Order relationship sets $\Theta$, $\Theta'$, the Split-Order distance between $T$ and $T'$ is defined as*

$$\begin{aligned} &SODist(T, T') = \\ &|\{SplitOrder(\sigma_i, \sigma_j, \sigma_k) \ s.t. \ SplitOrder(\sigma_i, \sigma_j, \sigma_k) \neq SplitOrder(\sigma'_i, \sigma'_j, \sigma'_k)\}| \end{aligned} \tag{5}$$

*where $(SplitOrder(\sigma_i, \sigma_j, \sigma_k) = \theta_{i,j,k}) \in \Theta$, and $(SplitOrder(\sigma'_i, \sigma'_j, \sigma'_k) = \theta'_{i,j,k}) \in \Theta'$.*

*The relative Split-Order distance is defined as:*

$$SODistRel(T, T') = SODist(T, T')/n^3 \tag{6}$$

**Algorithm 1.** ReconstructTree($\Sigma$,$\Theta$)
**Input** $\Sigma$: the leaf label set; $\Theta$: the set of Split-Order relations over $\Sigma$, $\Theta = \{SplitOrder(\sigma_i, \sigma_j, \sigma_k) = \theta_{i,j,k}\}, \forall \sigma_i, \sigma_j, \sigma_k \in \Sigma$, $\theta_{i,j,k} \in \{\prec, \succ, =\}$

```
 1: $\Sigma_{curr} \leftarrow \Sigma$
 2: while  do
 3:   if $|\Sigma_{curr}| = 1$ then
 4:     The only node in $\Sigma_{curr}$ is the root, return;
 5:   else
 6:     while $\Sigma_{curr}$ contains unprocessed nodes do
 7:       Pick an unprocessed node $\sigma_i \in \Sigma_{curr}$.
 8:       Find the set of nodes $\{\sigma_j\}$ which have the latest split with $\sigma_i$, according to
          Equation 1.
 9:       Test conflict, according to Equations 2 and 3.
10:       if there is a conflict then
11:         Tree does not exist, return
12:       else
13:         Create a new node $\sigma_{new}$ as the parent of $\sigma_i$ and all nodes in $\{\sigma_j\}$
14:         Add $\sigma_{new}$ to $\Sigma_{next}$
15:         Mark $\sigma_i$ and all nodes in $\{\sigma_j\}$ as processed
16:       end if
17:     end while
18:     $\Sigma_{curr} \leftarrow \Sigma_{next}$
19:   end if
20: end while
```

*where* $n = |\Sigma|$. *In other words,* $n^3 = |\Theta| = |\Theta'|$. *Note that* $SODistRel(T, T') \in [0, 1]$.

**Theorem 2.** *The Split-Order distance is a metric.*

*Proof.* Assume that we have three rooted, unordered trees $T_1$, $T_2$, $T_3$ with the same set of leaf labels $\Sigma$. The proof for symmetry, identity and non-negativity properties are intuitive and omitted here. We will prove the triangle inequality: $SODist(T_1, T_2) \leq SODist(T_1, T_3) + SODist(T_2, T_3)$. Denote the total number of Split-Order relations for each tree as $m$. Then the number of common Split-Order relations shared between $T_1$ and $T_3$ is $m - SODist(T_1, T_3)$, and the number of common Split-Order relations shared between $T_2$ and $T_3$ is $m - SODist(T_2, T_3)$. Therefore, the number of common Split-Order relations shared between $T_1$, $T_2$, and $T_3$ is at least

$$\begin{aligned}&(m - SODist(T_1, T_3)) + (m - SODist(T_2, T_3)) - m\\&= m - (SODist(T_1, T_3) + SODist(T_2, T_3))\end{aligned}$$

Thus, the number of Split-Order relations which are different between $T_1$ and $T_2$ is at most

$$\begin{aligned}&m - (m - (SODist(T_1, T_3) + SODist(T_2, T_3)))\\&= SODist(T_1, T_3) + SODist(T_2, T_3)\end{aligned}$$

## 5 Split-Order Distance Computation

We propose an efficient algorithm for computing the Split-Order distance between any two trees $T_1$, $T_2$ over the same set of leaf objects $\Sigma$.

The naive algorithm computes the complete set of the Split-Order relations for both trees and counts the number of different Split-Order relations in two trees. Since there are $O(n^3)$ Split-Order relations for each tree, counting the different Split-Order relations alone will take $O(n^3)$ time. In fact, computing a single Split-Order relation has more than $O(1)$ complexity. In the following discussion, we propose an efficient algorithm for computing Split-Order distance which takes only $O(n^2 d^4)$ time, where $d$ is the maximum degree of any node.

Let $\Theta(T)$ denote the complete set of the Split-Order relations of tree $T$. For each internal node $n_p$, we compute a subset of $\Theta(T)$, denoted as $SplitOrderSet(T, n_p)$:

$$\begin{aligned}&SplitOrderSet(T, n_p) = \{SplitOrder(\sigma_{i_1}, \sigma_{i_2}, \sigma_{i_3}) = \theta_{i_1,i_2,i_3} \ where\\ &(Split(\sigma_{i_1}, \sigma_{i_2}) = n_p) \wedge ((SplitOrder(\sigma_{i_1}, \sigma_{i_2}, \sigma_{i_3}) = \theta_{i_1,i_2,i_3}) \in \Theta(T))\}\end{aligned} \tag{7}$$

It is easy to prove that $\Theta(T)$ can be divided into disjoint sets of $SplitOrderSet(T, n_p)$:

$$\cup_{n_p} SplitOrderSet(T, n_p) = \Theta(T) \tag{8}$$

$$SplitOrderSet(T, n_{p_1}) \cap SplitOrderSet(T, n_{p_2}) = \phi \tag{9}$$

The basic idea of our Split-Order distance computation algorithm is to count, for any internal node $n_p$ in $T$ and any internal node $n_{p'}$ in $T'$, the number of common Split-Order relations associated with them, *i.e.*, $|SplitOrderSet(T, n_p) \cap SplitOrderSet(T', n_{p'})|$. In the following discussion, we only consider the Split-Order relations of type '$\succ$' and '$=$', since we have

$$\begin{aligned}&SplitOrder(\sigma_{i_1}, \sigma_{i_2}, \sigma_{i_3}) =' \prec' \Leftrightarrow\\ &SplitOrder(\sigma_{i_1}, \sigma_{i_3}, \sigma_{i_2}) =' \succ'\end{aligned} \tag{10}$$

The total number of common Split-Order relations is twice the number of total common '$\succ$'-type Split-Order relations plus the number of total common '$=$'-type Split-Order relations. Furthermore, we only consider Split-Order relations where $\sigma_{i_1}$, $\sigma_{i_2}$, and $\sigma_{i_3}$ are three different leaf labels. If any two of them are the same we already know that the relation is common between $\Theta(T)$ and $\Theta(T')$.

We first explain how to compute the Split-Order relations of type '$\succ$' and '$=$' in $SplitOrderSet(T, n_p)$ given a tree $T$ and an internal node $n_p$. Denote the set of leaf nodes which are inside the subtree rooted at $n_p$ as $Leaves(n_p)$, and the $k^{th}$ child of $n_p$ as $Child(n_p, k)$ (see Fig. 4). It is easy to prove that $Leaves(n_p) = \cup_k Leaves(Child(n_p, k))$, and $Leaves(Child(n_p, k)) \cap Leaves(Child(n_p, k')) = \phi$. Here we assume that all child nodes are in an ordered list for the convenience of discussion. The child nodes can be of any order. For example in Fig. 3, consider $n_4$, $Leaves(n_4) = \{a, b, c\}$. $n_9$ and $n_{10}$ are the children of $n_4$, and $Leaves(n_9) = \{a, b\}$, $Leaves(n_{10}) = \{c\}$. Now we determine the necessary conditions for $SplitOrder(\sigma_{i_1}, \sigma_{i_2}, \sigma_{i_3})$ to be of '$\succ$' or '$=$' type. If $\sigma_{i_1} \in Leaves(Child(n_p, k_1))$, and $\sigma_{i_2} \in Leaves(Child(n_p, k_2))$, we know that $k_1 \neq k_2$. Otherwise, $Split(\sigma_{i_1}, \sigma_{i_2})$ is $Child(n_p, k_1)$ or a descendent of $Child(n_p, k_1)$, but not $n_p$. Therefore, $\sigma_{i_1}$,$\sigma_{i_2}$ are from different child nodes of $n_p$. For the requirement of $\sigma_{i_3}$, we have $\sigma_{i_3} \in \Sigma \backslash Leaves(Child(n_p, k_1))$. Otherwise, if $\sigma_{i_3} \in Leaves(Child(n_p, k_1))$, we have $Split(\sigma_{i_1}, \sigma_{i_2}, \sigma_{i_3}) =$'$\prec$'. Furthermore, depending on the type of the Split-Order relationship ('$\succ$' or '$=$'), $\sigma_{i_3}$ should belong to either $\Sigma \backslash Leaves(n_p)$ or $Leaves(n_p) \backslash Leaves(Child(n_p, k_1))$. In summary, $\sigma_{i_1}$, $\sigma_{i_2}$, $\sigma_{i_3}$ in a '$\succ$'-type Split-Order relation in $SplitOrderSet(T, n_p)$ should satisfy (see Fig. 4):

$$\begin{cases} \sigma_{i_1} \in Leaves(Child(n_p, k_1)) \\ \sigma_{i_2} \in Leaves(Child(n_p, k_2)) \wedge k_1 \neq k_2 \\ \sigma_{i_3} \in \Sigma \backslash Leaves(n_p) \end{cases}$$

where $1 \leq k_1, k_2 \leq K$, and $K$ is the number of children of node $n_p$. $\sigma_{i_1}$, $\sigma_{i_2}$, $\sigma_{i_3}$ in an '$=$'-type $SplitOrder$ relation in $SplitOrderSet(T, n_p)$ should satisfy (see Fig. 4):

$$\begin{cases} \sigma_{i_1} \in Leaves(Child(n_p, k_1)) \\ \sigma_{i_2} \in Leaves(Child(n_p, k_2)) \wedge k_1 \neq k_2 \\ \sigma_{i_3} \in Leaves(Child(n_p, k_3)) \wedge k_1 \neq k_3 \wedge \sigma_{i_2} \neq \sigma_{i_3} \end{cases}$$

where $1 \leq k_1, k_2 \leq K$, and $K$ is the number of children of $n_p$. As an example, for node $n_7$ in Fig. 3, we have $SplitOrder(f, h, i) =$'$=$' and $SplitOrder(f, h, a) =$'$\succ$'.

Now we explain how to compute the size of $SplitOrderSet(T, n_p) \cap SplitOrderSet\ (T', n_{p'})$ for a pair of internal nodes $n_p$, $n_{p'}$ in tree $T$ and $T'$, respectively. As discussed earlier, we consider the '$\succ$' and '$=$' Split-Order relations. The number of common '$\succ$' Split-Order relations $Shared_{\succ}(n_p, n_{p'})$ can be computed as follows:

$$\begin{aligned} &Shared_{\succ}(n_p, n_{p'}) = \Sigma_{1 \leq k_1 \neq k_2 \leq K, 1 \leq k'_1 \neq k'_2 \leq K'} \\ &(|Leaves(Child(n_p, k_1)) \cap Leaves(Child(n_{p'}, k'_1))| \\ &\times |Leaves(Child(n_p, k_2)) \cap Leaves(Child(n_{p'}, k'_2))| \\ &\times |(\Sigma \backslash Leaves(n_p)) \cap (\Sigma \backslash Leaves(n_{p'}))|) \end{aligned} \tag{11}$$

In both trees, a '$\succ$'-type Split-Order relation satisfies the following conditions: $\sigma_{i_1}$, $\sigma_{i_2}$ are descendants of two different children of $n_p$ and also of two different children of $n_{p'}$. In addition, $\sigma_{i_3}$ is not in either $Leaves(n_p)$ or $Leaves(n_{p'})$ (see Fig. 4). Assume that computing the size of a set intersection takes constant time (we will explain later how to compute it in constant time). Let the maximum number of children for a node be $d$. The computation of $Shared_{\succ}(n_p, n_{p'})$ takes $O(d^4)$ time.

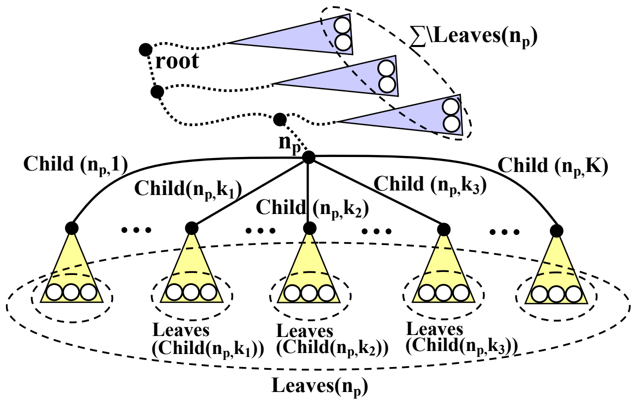


**Fig. 4.** Illustration of Split-Order distance computation

The number of common '='-type Split-Order relations $Shared_{=}(n_p, n_{p'})$ can be computed as follows:

$$\begin{aligned}&Shared_{=}(n_p, n_{p'}) = \Sigma_{1\le k_1\le K, 1\le k'_1\le K'}\\&(|Leaves(Child(n_p, k_1)) \cap Leaves(Child(n_{p'}, k'_1))|\\&\times(\Sigma_{1\le k_2\le K, 1\le k'_2\le K', k_2\ne k_1, k'_2\ne k'_1}\\&|Leaves(Child(n_p, k_1)) \cap Leaves(Child(n_{p'}, k'_2))|)\\&\times((\Sigma_{1\le k_2\le K, 1\le k'_2\le K', k_2\ne k_1, k'_2\ne k'_1}\\&|Leaves(Child(n_p, k_1)) \cap Leaves(Child(n_{p'}, k'_2))|) - 1))\end{aligned} \tag{12}$$

Similarly, in either tree, an '='-type Split-Order relation satisfies the following conditions: $\sigma_{i_1}$, $\sigma_{i_2}$, $\sigma_{i_3}$ are all descendants of $n_p$ and children of $n'_p$; also, $\sigma_{i_2}$ and $\sigma_{i_3}$ must be descended from children which are not ancestors of $\sigma_{i_1}$; finally, $\sigma_{i_2}$ are $\sigma_{i_3}$ are different leaf labels (see Fig. 4). Again, if a set intersection computation takes constant time, the computation of $Shared_{=}(n_p, n_{p'})$ takes $O(d^4)$ time.

Therefore, let the total number of common '$\succ$'-type Split-Order relations shared between two trees be $Shared_{\succ}(T, T')$, and the total number of common '='-type Split-Order relations shared between two trees be $Shared_{=}(T, T')$. Then we have:

$$Shared_{\succ}(T, T') = \Sigma_{\forall n_p\in T, n_{p'}\in T'} Shared_{\succ}(n_p, n_{p'}) \tag{13}$$

$$Shared_{=}(T, T') = \Sigma_{\forall n_p\in T, n_{p'}\in T'} Shared_{=}(n_p, n_{p'}) \tag{14}$$

Thus, the total number of common Split-Order relations shared between *SplitOrderSet*$(T, n_p)$ and *SplitOrderSet*$(T', n_{p'})$ is:

$$Shared(T, T') = Shared_{=}(T, T') + 2 \times Shared_{\succ}(T, T') \tag{15}$$

Therefore, the Split-Order distance between $T$ and $T'$ is:

$$SODist(T, T') = n^3 - (n + 3n(n-1)) - Shared(T, T') \tag{16}$$

Here $n^3$ is the total number of Split-Order relations for a leaf label set of size $n$, and $n+3n(n-1)$ is the number of Split-Order relations of which at least two of $\sigma_{i_1}$,$\sigma_{i_2}$,$\sigma_{i_3}$ have the same label. As mentioned before, these Split-Order relations must be common for both trees.

Now we explain how we can compute the size of the set intersection in constant time. The basic idea is that we compute in advance $|Leaves(n_p) \cap Leaves(n_{p'})|$ for all possible pairs of $n_p$ and $n_{p'}$, which are the internal nodes of $T$ and $T'$, respectively. This can be done in $O(n^2)$ time as follows:

1. Initially, we compute $|Leaves(n_p) \cap Leaves(n_{p'})|$ where $n_p$ and $n_{p'}$ are leaf nodes in $T$ and $T'$, respectively. Each $|Leaves(n_p) \cap Leaves(n_{p'})|$ can be done in constant time.
2. We compute $|Leaves(n_p) \cap Leaves(n_{p'})|$ if either of the following cases holds:
   (a) $|Leaves(Child(n_p, k)) \cap Leaves(n_{p'})|$ has been computed for all children of $n_p$
   (b) $|Leaves(n_p) \cap Leaves(Child(n_{p'}, k'))|$ has been computed for all children of $n_{p'}$

   This can be done in $O(d)$ time where $d$ is the maximum degree of any node. For case (a):
$$\begin{aligned}&|Leaves(Child(n_p, k)) \cap Leaves(n_{p'})| = \\ &\Sigma_k |Leaves(Child(n_p, k)) \cap Leaves(n_{p'})|\end{aligned} \tag{17}$$
   For case (b):
$$\begin{aligned}&|Leaves(Child(n_p, k)) \cap Leaves(n_{p'})| = \\ &\Sigma_{k'} |Leaves(n_p) \cap Leaves(Child(n_{p'}, k'))|\end{aligned} \tag{18}$$
3. Repeat 2) until $|Leaves(n_p) \cap Leaves(n_{p'})|$ is computed for any pair of nodes $n_p \in T$, and $n_{p'} \in T'$

Therefore, computing $Leaves(n_p) \cap Leaves(n_{p'})$ for all possible pairs of $n_p$ and $n_{p'}$ takes $O(dn^2)$ time. Note that in the computation of $Shared_{\succ}(n_p, n_{p'})$, we also need to compute $|(\Sigma \backslash Leaves(n_p)) \cap (\Sigma \backslash Leaves(n_{p'}))|$, which can be computed in constant time as follows:
$$\begin{aligned}&|(\Sigma \backslash Leaves(n_p)) \cap (\Sigma \backslash Leaves(n_{p'}))| = \\ &n - |Leaves(n_p)| - |Leaves(n_{p'})| + \\ &|Leaves(n_p) \cap Leaves(n_{p'})|\end{aligned} \tag{19}$$

We can easily compute $|Leaves(n_p)|$ and $|Leaves(n_{p'})|$ in advance for all $n_p \in T$ and $n_{p'} \in T'$ using a post-order traversal for both trees, which takes $O(nd)$ for each tree.

The complete algorithm for computing Split-Order distance is in Algorithm 2.

**Theorem 3.** *Algorithm 2 runs in $O(n^2d^4)$ time.*

*Proof.* To compute $SODist(T, T')$, we consider every pair of nodes $(n_p, n_{p'})$. The number of internal nodes for both trees are $O(n)$, therefore, we have $O(n^2)$ node pairs. Since computing $Shared(n_p, n_{p'})$ takes $O(d^4)$ time, the total time complexity of Algorithm 2 is $O(n^2d^4)$.

**Algorithm 2.** SODist($T$,$T'$,$\Sigma$)
**Input** $\Sigma$: the leaf label set; $T, T'$: two trees

```
1: Compute all possible set intersections: $|Leaves(n_p) \cap Leaves(n_{p'})|$, for any internal node $n_p \in T$ and any internal node $n_{p'} \in T'$
2: Compute all $|Leaves(n_p)|$, $|Leaves(n_{p'})|$, for any internal node $n_p \in T$ and any internal node $n_{p'} \in T'$
3: for any internal node $n_p \in T$ and any internal node $n_{p'} \in T'$ do
4:     Compute $Shared_{\succ}(n_p, n_{p'})$ and $Shared_{=}(n_p, n_{p'})$ according to Equation 10 and 11
5: end for
6: Compute $SODist(T, T')$ according to Equation 12,13,14 and 15.
```

## 6 Experimental Results

We test the performance of our algorithm on both real and synthetic data sets.

– **Iyer's Data[7]:** Iyer's Data contains gene expression levels of 517 human genes in response to serum stimulation over 12 time points. The 517 genes can be clustered into hierarchical structures based on their co-expressions [7,8].
– **Synthetic Data:** 1) **SYN-Random:** The synthetic dataset contains a set of randomly generated, rooted, leaf labeled, unordered trees. The simulation is controlled by two parameters: the number of leaves $n$, and the maximum degree $d$ of any internal node. 2) **SYN-2D:** The 2-D synthetic dataset contains 1000 sampled pixels from 4 shapes in an image, including 1% of background noises. Clustering hierarchy for SYN-2D is obtained by applying CLUTO software[1]. The 4 clusters (representing the four big branches in the hierarchy) are illustrated using different colors in Fig. 6(a).

We compared the performance of our algorithm against the tree edit distance. We used a Java implementation of Zhang-Shasha's tree edit distance algorithm [21] available online[2]. Zhang-Shasha's algorithm applies to ordered, labeled trees. Our Split-Order algorithm is implemented in both C++ and Java, and the experiments are performed on an Intel Core 2 Duo 1.6GHz machine with 3GB memory. The Java version of the Split-Order algorithm is mainly used for the comparison of running time performance with the Java implementation of Zhang-Shasha's tree edit distance algorithm.

### 6.1 Distance Evaluation

**Distance Comparison on Iyer's Data.** Iyer's Data contains gene expression levels of 517 human genes over 12 time points. We refer to the complete Iyer's Data as Iyer12. Six additional data sets are generated from Iyer12 by randomly

[1] http://glaros.dtc.umn.edu/gkhome/views/cluto
[2] http://www.ics.mq.edu.au/ swan/howtos/treedistance/package.html

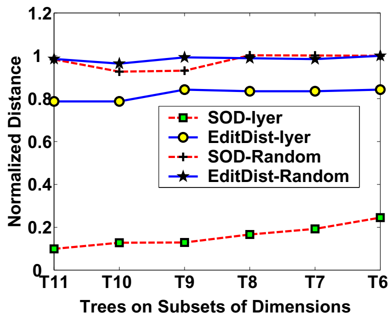


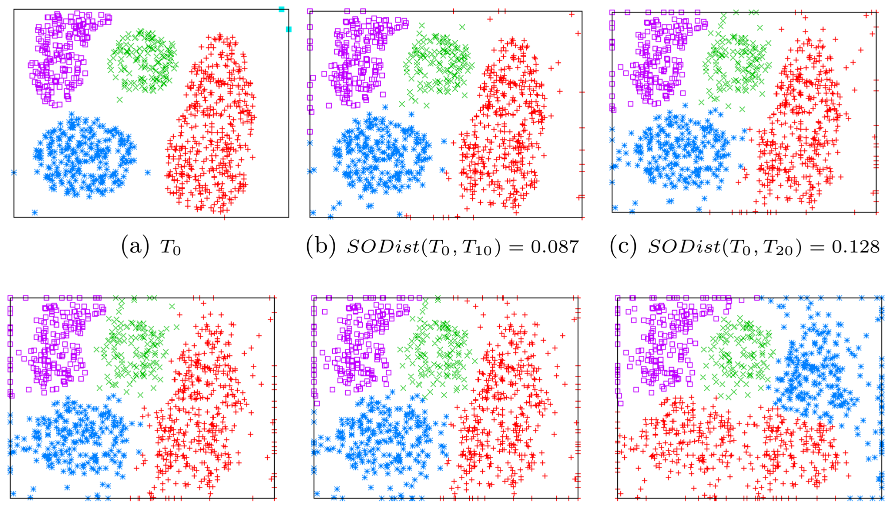
**Fig. 5.** Comparison of Split-Order distance and tree edit distance (Zhang-Shasha) on Iyer's data

choosing 11, 10, 9, 8, 7, 6 dimensions respectively. These data sets are referred to as Iyer11, Iyer10, Iyer9, Iyer8, Iyer7, and Iyer6. We applied the same hierarchical clustering algorithm on these 7 data sets with Euclidean distance and single linkage, and obtained 7 trees: $T_{12}$, $T_{11}$, $T_{10}$, $T_9$, $T_8$, $T_7$, and $T_6$. As more dimensions are removed, the gene correlation captured by the resulting tree differs more from the original tree $T_{12}$. We therefore expect that the distances of $T_{11}$, $T_{10}$, $T_9$, $T_8$, $T_7$, and $T_6$ to $T_{12}$ are in an increasing order.

Fig.5 compares the distance scores computed using Split-Order distance and tree edit distance on the Iyer's Data. The Split-Order scores are normalized with the Split-Order distance of a random tree to $T_{12}$ which is 0.664. Similarly, the tree edit distance scores are normalized with the tree edit distance between a random tree and $T_{12}$ which is 1433. The Split-Order distance curve demonstrates a clear increasing trend for distances of $T_{11}$, $T_{10}$, $T_9$, $T_8$, $T_7$, and $T_6$ to the original tree $T_{12}$, which is consistent with our expectation. Compared to tree edit distance, Split-Order distance generates a much smaller distance for similar trees. The Split-Order distance between $T_{11}$ and $T_{12}$ is 10% of the distance between a random tree and $T_{12}$. The tree edit distance between $T_{11}$ and $T_{12}$ is however 80% of the distance between a random tree and $T_{12}$.

We also generated a random matrix of the same size (517 by 12) and performed the same experiment on this data. The corresponding curves are plotted in Fig.5. The two curves on random data are flat. The different behaviors of the random data curves and the Iyer's data curves can be explained by the gene correlation existing in the Iyer's data and the captured tree similarity due to the data correlation.

**Distance Evaluation on Synthetic Data.** We performed the distance evaluation on clustering hierarchies generated using SYN-2D. SYN-2D contains 1000 pixels uniformly sampled from 4 different shapes in an image, including 1% of background noise. We generated 5 additional data sets by randomly choosing 10%, 20%, 30%, 40%, and 50% of pixels in SYN-2D and applied a random perturbation on each pixel. Clustering hierarchies are generated on all 6 data sets using the hierarchical clustering function equipped with CLUTO [9]. We refer to



(a) $T_0$ (b) $SODist(T_0, T_{10}) = 0.087$ (c) $SODist(T_0, T_{20}) = 0.128$

(d) $SODist(T_0, T_{30}) = 0.176$ (e) $SODist(T_0, T_{40}) = 0.217$ (f) $SODist(T_0, T_{50}) = 0.458$

**Fig. 6.** Illustration of 4 big clusters in the clustering hierarchies computed for SYN-2D and its perturbed data sets. The 4 clusters are represented using squares(in purple), crosses(in green), stars(in blue) and pluses(in red). $T_0$ is the clustering hierarchy computed on SYN-2D, $T_{10}$, $T_{20}$, $T_{30}$, $T_{40}$, $T_{50}$ are the clustering hierarchies computed on SYN-2D with 10%, 20%, 30%, 40%, 50% of points applied perturbation.

the corresponding trees as $T_0$, $T_{10}$, $T_{20}$, $T_{30}$, $T_{40}$, and $T_{50}$. The four big clusters (the four big branches) in each tree are plotted in Fig.6(a)-Fig.6(f).

The Split-Order distances of each of $T_0$ to $T_{10}$, $T_{20}$, $T_{30}$, $T_{40}$, and $T_{50}$ are 0.087, 0.128, 0.176, 0.216, and 0.458, respectively. The visual differences among Fig.6(a)-Fig.6(f) reflect the clustering hierarchy differences among the corresponding trees. The increasing order of the Split-Order differences are accordant with the increasing visual differences shown in Fig.6(b) to Fig.6(f). Particularly, from $T_{50}$ to $T_{40}$, the Split-Order distance increases dramatically by more than 100%. Comparing Fig. 6(e) and Fig. 6(f), we also observe a big change: the blue cluster (stars) in 6(e) becomes part of the red cluster (pluses) in 6(f); part of the red cluster (pluses) in 6(e) becomes blue cluster (stars) in 6(f). This implies that the big branch representing the blue cluster in $T_{40}$ switches positions with a subbranch of the red cluster. This change impacts the relationship between a large number of pixels, which results in a big difference in the corresponding Split-Order distances.

### 6.2 Running Time Performance

We compare the running time performance of our algorithm (the Java version) with tree edit distance (the Java implementation of Zhang-Shasha algorithm) on the SYN-Random dataset.

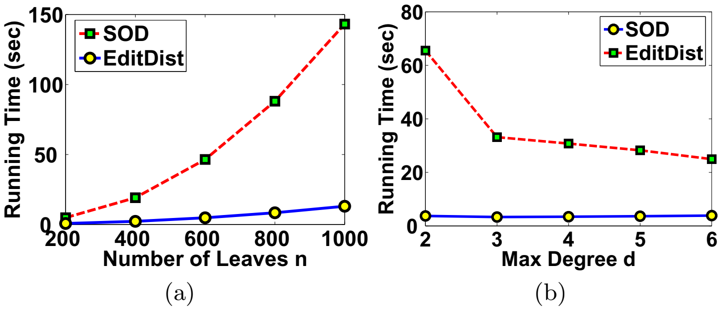


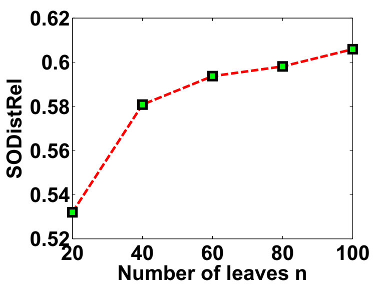
**Fig. 7.** Running time comparison of Split-Order distance (the Java implementation) and tree edit distance (the Java implementation of Zhang-Shasha) with varying number of leaf nodes $n$ ((a)) and with varying max degree $d$ ((b))

**Running time vs. $n$:** Fig. 7(a) compares the running time of Split-Order and Zhang-Shasha on trees with varying number of leaves $n$. As $n$ increases from 200 to 1000, the running time for both algorithms increases. The Split-Order distance is up to 10 times faster than the Zhang-Shasha algorithm and demonstrates better scalability with increasing $n$. Each data point is the average time taken for 10 distance computations for a given $n$. Although the running time of Zhang-Shasha is acceptable for reasonably large trees (less than 150 seconds for trees with 1000 leaves), the speedup is still necessary considering large number of tree edit distance computations required for applications such as querying tree databases.

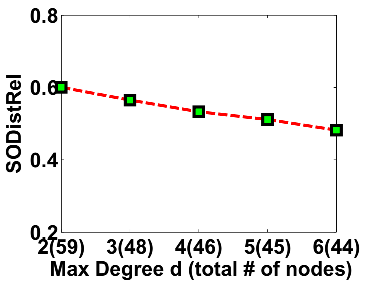
**Running time vs. $d$:** Fig. 7(b) compares the running time of Split-Order and Zhang-Shasha on trees with varying maximum degree $d$. The tree edit distance demonstrates a clear decline with increasing $d$, due to reduced total number of nodes with increasing $d$ and fixed number of leaves $n$. The Split-Order curve remains almost flat with small variations. The time complexity of our algorithm depends on the number of internal nodes (which is $O(n)$) and the maximum degree of any node $d$. The running time increases with larger $d$. However, with fixed $n$ and increasing $d$, the number of internal nodes decreases, which cancels out the increase introduced by larger $d$. The data is averaged over 10 runs of the algorithms.

## 6.3 *SODist* Distribution

We examine the distribution of *SODist* between any two trees using SYN-Random. 100 pairs of random trees are chosen and the relative Split-Order distances (*SODistRel*) are computed for each pair. Fig. 8(a) and 8(b) illustrate the distribution of the average *SODistRel* with varying parameters. We observe that the average *SODistRel* between any two random trees roughly falls within the range [0.5, 0.6]. There are small variations when parameters change. As shown in Fig. 8(a), *SODistRel* ranges between 0.53 and 0.61 as $n$ varies from

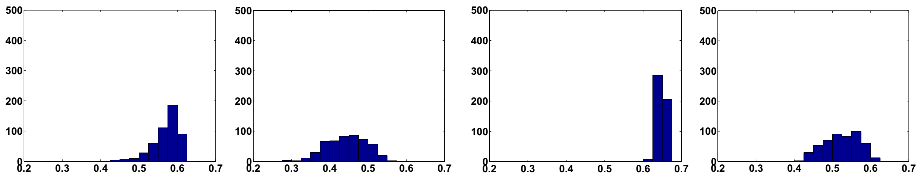


(a) *Average SODistRel vs. n*

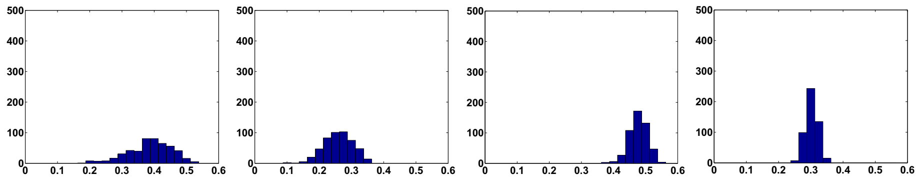


(b) *Average SODistRel vs. d*

**Fig. 8.** The average *SODistRel* between random trees with varying parameters: $n$ (the number of leaf nodes) and $d$ (the maximum number of children an internal node could have in a tree). On the X axis of (b), the average total number of nodes for the trees with $d$ is also given.



(a) $n = 20,d = 2$ (b) $n = 20,\ d = 6$ (c) $n = 100,\ d = 2$ (d) $n = 100,\ d = 6$



(e) $n = 20,d = 2$ (f) $n = 20,\ d = 6$ (g) $n = 100,\ d = 2$ (h) $n = 100,\ d = 6$

**Fig. 9.** The distribution of *SODistRel* and tree edit distance between random trees in different settings. A total number of 500 Split-Order distance computations are performed. X axis is *SODistRel* or tree edit distance, Y axis is the number of the distance computations contained in each bin.

20 to 100 and $d$ is fixed to be 3. The increase in the average *SODistRel* between random trees is due to the increased topological variety of trees when $n$ is larger. In Fig. 8(b), the average *SODistRel* between random trees is computed as $d$ varies from 2 to 6, and $n$ is fixed to be 30. We can observe a slow drop in the average *SODistRel* between random trees when $d$ increases. This is due to the decrease in the number of internal nodes with increasing $d$ and fixed $n$. As tree becomes flatter, the amount of topological variety of trees also decreases.

Fig. 9(a)-9(d) plots the histogram of 500 *SODistRel*s between 500 pairs of random trees with different parameters settings: $n = 20$ and $d = 2$ (Fig. 9(a)), $n = 20$ and $d = 6$ (Fig. 9(b)), $n = 100$ and $d = 2$ (Fig. 9(c)), $n = 100$ and

$d = 6$ (Fig. 9(d)). We observe that all distributions have the bell shape. For larger $d$ ($d = 6$ compared to $d = 2$), more diversity exists in the tree space and the histogram is more symmetric. For larger $n$ ($n = 100$ compared to $n = 20$), *SODistRel* becomes larger, and the mean of the distribution moves further to the right. These histograms depict the variation of the tree space when parameters change evaluated by the Split-Order distance.

For comparison, we plot the four corresponding histograms of tree edit distances computed over the same 500 pairs of random trees in Figs. 9(e), 9(f),9(g) and 9(h). The edit distance scores are normalized by $4n$ to get the relative scores. $4n$ is the upper bound of the tree edit distance between two trees with $n$ leaves. We observe that when $d$ increases (from 2 to 6), the variance of the tree edit distance declines. This contradicts the fact that the topological diversity increases as $d$ increases, which suggests that the tree edit distance is not an ideal measure.

## 7 Conclusion and Future Work

In this paper, we proposed a novel metric, Split-Order distance, to evaluate the distance between two clustering or classification hierarchies. This metric compares the order of the splits in both trees. We proved that a complete set of the split order relations uniquely determines the hierarchical structure. We also proposed an algorithm for reconstructing the tree using the set of order relations. Furthermore, we presented an efficient algorithm for computing the Split-Order distance between two hierarchies which takes $O(n^2d^4)$ time, where $n$ is the number of leaf nodes (objects), and $d$ is the maximum number of children of an internal node in a tree.

In the future, we would like to take weighted edges into consideration where weights represent the distance from a node to its parent. This allows us to model the situations where a single edge with different weights can result in different relationships among objects, such as in a dendrogram with different edge lengths.

## Acknowledgement

This work was partially supported by NSF grants IIS-0448392 and IIS-0812464.

## References

1. Allen, B.L., Steel, M.: Subtree transfer operations and their induced metrics on evolutionary trees. Annals of Combinatorics 5, 1–13 (2001)
2. Amir, A., Keselman, D.: Maximum agreement subtree in a set of evolutionary trees: metrics and efficient algorithms. Proc. of the SIAM Journal on Computing 26(6), 1656–1669 (1997)
3. Bille, P.: A survey on tree edit distance and related problems. Theoretical Computer Science, 217–239 (2005)
4. Demaine, E.D., Mozes, S., Rossman, B., Weimann, O.: An Optimal Decomposition Algorithm for Tree Edit Distance. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 146–157. Springer, Heidelberg (2007)

5. Eastabrook, G.F., McMorris, F.R., Meacham, C.A.: Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units. Syst. Zool (1985)
6. Felsenstein, J.: Cases in which parsimony and compatibility methods will be positively misleading. Syst. Zool. 27, 401–410 (1978)
7. Iyer, V.R., Eisen, M.B., Ross, D.T., Schuler, G., Moore, T., Lee, J.C., Trent, J.M., Staudt, L.M., Hudson Jr., J., Boguski, M.S., Lashkari, D., Shalon, D., Botstein, D., Brown, P.O.: Transcriptional program in the response of human fibroblasts to serum. Science 283, 83–87 (1996)
8. Jiang, D., Pei, J., Zhang, A.: DHC: a density-based hierarchical clustering method for time series gene expression data. In: Proc. of the The Third Symposium on Bioinformatics and Bioengineering, pp. 393–400 (2003)
9. Karypis, G.: CLUTO - A Clustering Toolkit. Tech Report, Dept. of Computer Science, University of Minnesota (2002)
10. Klein, P.N.: Computing the edit-distance between unrooted ordered trees. In: Bilardi, G., Pietracaprina, A., Italiano, G.F., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, p. 91. Springer, Heidelberg (1998)
11. Lu, S.Y.: A tree-to-tree distance and its application to cluster analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) (1979)
12. Robinson, D.F., Foulds, L.R.: Comparison of weighted labeled trees. Combinatorial mathematics VI, 119–126 (1979)
13. Robinson, D.F., Foulds, L.: Comparison of phylogenetic trees. Math. Biosci. 53(1-2), 131–147 (1981)
14. Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. The Mol. Biol. Evol. 4(4), 406–425 (1987)
15. Shasha, D., Wang, J.T.L., Zhang, S.: Unordered Tree Mining with Applications to Phylogeny. In: Proc. IEEE International Conference on Data Engineering (ICDE 2004) (2004)
16. Sneath, P.H.A., Sokal, R.R.: Numerical Taxonomy, pp. 230–234. WH Freeman and Company, New York (1973)
17. Touzet, H.: Tree edit distance with gaps. Information Processing Letters 85(3), 123–129 (2003)
18. Wang, J.T., Zhang, K., Jeong, K., Shasha, D.: A system for approximate tree matching. IEEE Transactions on Knowledge and Data Engineering 6(4), 559–571 (1994)
19. Eastabrook, G.F., McMorris, F.R., Meacham, C.A.: TreeRank: A similarity measure for nearest neighbor searching in phylogenetic databases. In: Proc. of the 15th International Conference on Scientific and Statistical Database Management (1985)
20. Waterman, M.S., Smith, T.F.: On the similarity of dendrograms. Journal of Theoretical Biology 73, 789–800 (1978)
21. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM Journal of Computing (1989)

# Combining Multiple Interrelated Streams for Incremental Clustering

Zaigham Faraz Siddiqui and Myra Spiliopoulou

Otto-von-Guericke-University of Magdeburg,
Magdeburg 39106, Germany
{siddiqui,myra}@iti.cs.uni-magdeburg.de

**Abstract.** Many data mining applications analyze structured data that span across many tables *and* accumulate in time. Incremental mining methods have been devised to adapt patterns to new tuples. However, they have been designed for data in one table only. We propose a method for incremental clustering on multiple interrelated streams - a "*multi-table stream*": its components are streams that reference each other, arrive at different speeds and have attributes of a priori unknown value ranges. Our approach encompasses solutions for the maintenance of caches and sliding windows over the individual streams, the propagation of foreign keys across streams, the transformation of all streams into a single-table stream, and an incremental clustering algorithm that operates over that stream. We evaluate our method on two real datasets and show that it approximates well the performance of an ideal method that possesses unlimited resources and knows the future.

## 1 Introduction

Many knowledge discovery applications refer to data that span more than one tables. For example, a customer of an online store is described by her web sessions, the products she has purchased, the reclamations she has performed etc. If we want to build profiles by clustering similar customers, we must consider their purchases, reclamations and navigation habits, next to their personal data. Most mining techniques cannot deal with this problem, because they operate on single-table data. Methods that learn on multiple tables[1,2,3,4,5,6,7,8,9] deal with static data. Many applications require the analysis of stream data, though, such as the example above: there is a fast stream of transactions and a stream of reclamations, a slower stream of customers (new and recurring ones), a stream of products (that come in and out of the company's portfolio). We propose a method for clustering such a "multi-table stream" i.e. a set of tables that *reference each other* and accumulate or flow as *streams* of different speeds.

For conventional stream mining over a single data table [1], the stream can be perceived as a sequence of data points $x_1, \ldots, x_i, \ldots$; a window of length $L$ slides over them, and the the mining algorithm builds a model $\zeta$ on the tuples

[1] This outline is based on the "stream paradigm" described in [10].

within the window. When new tuples arrive, the oldest ones are forgotten and $\zeta$ is adapted to the updated content of the sliding window.

Multi-table stream mining cannot be easily fit into the above process. To mine a multi-table stream, we must first select the "target" stream $T_0$ that will be the basis of the model. If the online store of our example is interested in customer profiles, the target stream is that of the customers; if they rather want to identify trends in reclamations, the target stream is that of reclamations.

Once the target $T_0$ is specified, information from all other streams must be propagated to it. Assume that each tuple $z \in T_0$ joins with $n_z \geq 0$ tuples of stream $T$, e.g. the stream of purchases, the tuples of which join in turn with those of other streams. To build a model $\zeta$ over the $T_0$ tuples in the current sliding window, we must first join them with the corresponding $T$ tuples, which must also joined first with other streams. This implies that (a) we must specify how many $T_0$ tuples will be "cached" until the tuples to be joined with them arrive, (b) we must maintain one window for each stream $T$ that joins with $T_0$, each stream that joins with $T$ etc and (c) when a new tuple of $T_0$ arrives, we must wait for all windows to be updated before we adapt the model.

These requirements seem to call for an n-way join but this still not sufficient for mining. The reason is that mining algorithms assume that tuples are independent. This is violated whenever a tuple of the target joins with more than one tuples of another stream. In the above example, assume 10 elder and one teenager customers, and assume that the teenager made 10 purchases, while each other customer made only one purchase. The join result contains 21 tuples, which a mining algorithm will assume to be independent: it will consider the 10 tuples of the teenager as 10 independent teenagers. This will most likely lead to dubious conclusions about customer age distribution and buying preferences. Multi-relational mining methods solve this problem by turning rows to columns: instead of producing 10 tuples for the teenager, all her purchases are summarized in additional *columns* of *one* output tuple. In stream mining this is problematic, because the number of columns to be added to each stream tuple is not known a priori. Hence, multi-table stream mining also requires (d) a strategy that maintains the arriving data into a structure of fixed size.

Our approach deals with the above issues. We generalize the one-window scenario of stream mining into a scenario involving caches and windows over all components of the multi-table stream. We propose a mechanism that transforms the multi-table stream into a single-table stream containing exactly one tuple per original tuple of the target stream. We accompany this mechanism by a strategy that updates the caches and windows before model adaptation and heuristically minimizes information loss with respect to model learning. We couple our method with conventional stream clustering [11].

The paper is organized as follows. In Section 2 we discuss related work. In Section 3 we present our cache-and-window management strategy and the stream propositionalization mechanism that transforms a multi-table stream to a single stream. Our evaluation framework and experimental results are in Section 4. We conclude with a summary of our findings and future work.

## 2 Related Work

Clustering on a stream composed of multiple tables is a new problem. Related work comes from multi-table learning on static data and from single-table stream clustering. From the latter area, we depict solutions on referencing old data.

Stream mining algorithms process and then discard the arriving tuples. For example, Guha et al maintain at each moment the $m$ most recent tuples and $K$ medians that stand for $K \times m$ tuples seen in the past [10]. However, multi-table stream mining require the reconstruction of tuples that may refer to tuples seen much earlier. Tuple reconstruction is studied in the context of stream joins.

Stream join algorithms incorporate a mechanism that discards old tuples. `PROB` retains those tuples that are referenced many times [12], while `LIFE` discards the oldest ones [12]. `AGE` assumes that the benefit of storing a tuple is a function of its age, modeled as the time it remains within the sliding window [13]. The motivation for `AGE` comes for online auction sites where a tuple receives many bids just before it expires [13]. If a tuple is discarded and then other tuples reference it, then the result of the stream join is a subset of the complete result, achieved without forgetting. `PROB` attempts to maximize this subset [12], while `Archive Matrix` of `AGE` [13] aims to build a proper random sample rather than maximizing the subset of available join results.

Not all data referenced by a stream are themselves streams. Xie et al distinguish between joins involving two streams and those involving a stream and a static table [14]. The first operation is termed *joining*, the second one *caching.* Xie's caching differs from the conventional one, because a conventional cache miss is mended by retrieving the missing tuple. "In contrast, when caching stream tuples, a miss can cost a lot more: when a tuple is discarded, it is irrevocably gone along with all result tuples that it could generate in the future" [14]. In classic caching only the first reference is important; in stream joins, references beyond the first one are also important. Xie et al propose `HEEB`, a technique that maximizes the expected benefit of a fixed-size cache under the `MAX-subset` measure of `PROB` [14]. Unlike `PROB` and `LIFE` which are hardwired heuristics, `HEEB` exploits the statistical properties of the arriving data [14].

These techniques only consider joins between two streams or between a stream and a database table. Our cache-and-window management strategy deals with joins involving more than one streams when preparing the data for mining.

Multi-relational mining algorithms exploit the schema to deduce the order in which the tables must be processed to learn the patterns. Many of these methods are based on Inductive Logic Programming (ILP) [15]. They include multi-relational association rules [2,3], multi-relational decision trees [1,5] and distance measures [16] for clustering [17]. However, they have all been designed for static data and have high complexity [6].

Some multi-relational mining tools also deal with patterns that change over time[8,9]. The database mining system PANDA supports elaborate pattern comparison [8], while PSYCHO supports temporal modeling and updating of patterns [9]. None of them supports streams, though: pattern updating is initiated manually when the static data are considered to have changed substantially.

Multi-relational mining methods are dedicated algorithms that operate in a database. In contrast, *propositionalization* methods like [4,7,18,6] prepare static data as input to conventional mining algorithms. They transform static multi-table data into a single table that preserves the original semantics *and* alleviates the problem of interdependent tuples pointed out in Section 1. They do so by summarizing the contents of interdependent tuples in additional columns.

The early propositionalization approach LINUS [4] used logic programming as part of an ILP system. It was later extended to handle determinate clauses [7]. The propositionalization overhead grows exponentially with the number of new columns/attributes. The algorithms of Kobbe et al [18] and of Kroegel [6] rather use join and data aggregation to build the output table.

The overhead of `RelAggs` grows linearly to the number of additional columns [6]. To achieve this, `RelAggs` does not join all tables at once. It rather specifies a *target table*, to which all other tables are joined in a sequence of join steps, starting with those tables that are in relationship to the target table. Since the overhead of this process is $O(2^n)$ to the number of tables $n$, `RelAggs` also performs *target ID propagation*, i.e. it propagates the key attribute of the target table to all tables, thus transforming the original schema into a kind of a *star schema*[2] that connects all tables to the target table (see Fig. 3). In our stream propositionalization approach we adopt the join sequencing and target ID propagation of `RelAggs` to reduce the overhead of processing the input streams.

## 3 Multi-table Stream Propositionalization for Clustering

To prepare a multi-table stream for clustering, we first specify the *target* table/stream $T_0$. Mining is performed at timepoints $t_1, \ldots, t_i, \ldots$. At each $t_i$, a model $\zeta_i$ upon the current data of $T_0$ and of the other streams referring to it. In the stream paradigm [10], $t_i$ is the arrival time of tuple $x_i$. In our scenario, $t_i$ can be defined similarly, or it may be the end of some period, e.g. a week or year: this is closer to the intuition of building models at regular intervals. In subsection 3.1 we present the cache-and-window management strategy we use to collect the data of each stream needed to built $\zeta_i$ at each $t_i$ and explain how we update these memory management structures to prepare for the next model $\zeta_{i+1}$ at $t_{i+1}$. In subsection 3.2 we present the stream propositionalization algorithm that turns the multi-table stream at each $t_i$ into a single-table stream snapshot for mining. Table 1 summarizes our notation.

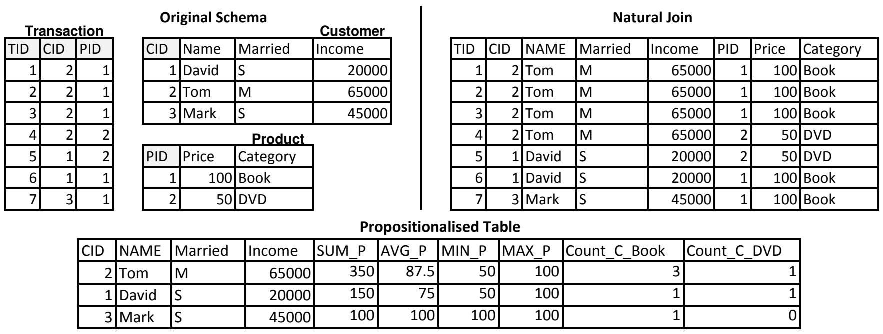
### 3.1 Cache-and-Window Management over a Multi-table Stream

In a multi-table stream, a tuple of the target table $T_0$ must be expanded with all relevant information before being processed and then forgotten. For example, consider the data in Fig. 1(a) and assume that `Customer` is the target table: a customer tuple must be joined with all transactions of this customer, and should

[2] Not to be confused with the "star schema" in OLAP.

**Table 1.** Symbols & Terms

| **Symbol** | **Description** |
|---|---|
| $T_j$ with schema $\mathcal{S}_j$ | $j^{th}$ comp. of the multi-table stream, $j = 0, \ldots, n$; $T_0$ is target |
| $C_j, W_j, Y_j$ | Cache XOR Window associated with $T_j$ - $Y_j$ stands for either |
| $\mathcal{X}^{\mathcal{B}}$ | Original schema of the data, composed of $\mathcal{S}_j, j = 0, \ldots, n$ and the relationships among them |
| $T \overset{\alpha}{\text{—}} T'$ | relationship b/w streams $T$, $T'$, $\alpha \in$ {1-to-1,1-to-N,M-to-N} |
| $x \rightarrow y$ | tuple $x \in T$ references tuple $y \in T'$ |
| $\mathcal{X}^{\mathcal{C}}$ | "Star" schema of the data, with $T_0$ in its center |
| $t_1, t_i$ | Initial and current timepoint |
| $\mathcal{S}_{i,0}$ | Schema of the propositionalized target stream $T_0$ at $t_i$ |
| $C_{i,j}, W_{i,j}, Y_{i,j}$ | contents of $C_j$, resp. $W_j$ at timepoint $t_i$ - $Y_{i,j}$ stands for either |
| $Q_{i,j}$ | Number of references to each referenced tuple of $T_j$ at $t_i$; $Q_{i,j}(x)$ refers to $x \in T_j$ |
| $domain(A, i)$ | Domain of a nominal attribute $A \in \cup_j \mathcal{S}_j$ at timepoint $t_i$ |
| $columns(A, i) \in [l_A, r_A]$ | Number of columns allocated to att. $A$ at $t_i$; ranges between max no. of permitted positions $r_A$ and a lower boundary $l_A$ |


Original Schema

Transaction

| TID | CID | PID |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 2 | 1 |
| 3 | 2 | 1 |
| 4 | 2 | 2 |
| 5 | 1 | 2 |
| 6 | 1 | 1 |
| 7 | 3 | 1 |

Customer

| CID | Name | Married | Income |
|---|---|---|---|
| 1 | David | S | 20000 |
| 2 | Tom | M | 65000 |
| 3 | Mark | S | 45000 |

Product

| PID | Price | Category |
|---|---|---|
| 1 | 100 | Book |
| 2 | 50 | DVD |

Natural Join

| TID | CID | NAME | Married | Income | PID | Price | Category |
|---|---|---|---|---|---|---|---|
| 1 | 2 | Tom | M | 65000 | 1 | 100 | Book |
| 2 | 2 | Tom | M | 65000 | 1 | 100 | Book |
| 3 | 2 | Tom | M | 65000 | 1 | 100 | Book |
| 4 | 2 | Tom | M | 65000 | 2 | 50 | DVD |
| 5 | 1 | David | S | 20000 | 2 | 50 | DVD |
| 6 | 1 | David | S | 20000 | 1 | 100 | Book |
| 7 | 3 | Mark | S | 45000 | 1 | 100 | Book |

Propositionalised Table

| CID | NAME | Married | Income | SUM_P | AVG_P | MIN_P | MAX_P | Count_C_Book | Count_C_DVD |
|---|---|---|---|---|---|---|---|---|---|
| 2 | Tom | M | 65000 | 350 | 87.5 | 50 | 100 | 3 | 1 |
| 1 | David | S | 20000 | 150 | 75 | 50 | 100 | 1 | 1 |
| 3 | Mark | S | 45000 | 100 | 100 | 100 | 100 | 1 | 0 |

**Fig. 1.** First tuples of (a) a multi-table stream on customers, transactions and products, (b) n-way join of the `Customer` data with `Transaction` and `Product` information and (c) propositionalized version of the same target `Customer`

be thus kept available for as long as such transactions are expected. On the other hand, a transaction tuple can be discarded immediately after being read. Hence, we distinguish between *tuples that may be forgotten* and *tuples that may not be forgotten*. We define a sliding window for the former and we use only those inside the window for processing. For tuples that may not be forgotten we use a cache, defined as in [14], as well as secondary storage for long-term maintenance [3].

When expanding a tuple $x \in T_0$ with tuples from another stream $T$, we may encounter the case that $x$ refers to already seen tuples $x_1, \ldots, x_m \in T$, not all of which are still in the cache. If such a tuple is in the cache, then it is *active*

[3] Long-term maintenance does not preclude tuple deletion. For example, an insurance company may delete a customer after her last contract has been paid out.

and is immediately joined to $x$. If it is not in the cache, it is *inactive*; it is then fetched from the secondary storage and joined to $x$ for the *next mining run*. It contributes to the next model $\zeta_{i+1}$. We next describe the cache updating strategy and explain the impact of tuples referencing other tuples.

**Cache-and-Window Updating.** We associate with each stream of tuples $T$ that may be forgotten a *time-based* sliding window $W_j$of length $L$: at timepoint $t_i$, it contains the tuples seen since $t_{i-L}$. A stream of tuples $T_{j'}$ that may not be forgotten is associated with a cache $C_{j'}$, which at first contains all tuples seen until the first timepoint $t_1$. Since the size of the cache is fixed, some tuples in window $W_{i,j}$ over stream $T_j$ at timepoint $t_i$ may refer to tuples of $T_{j'}$ that are not in $C_{i,j'}$ (cf. Table 1 for notation). These tuples must be fetched to the cache, discarding inactive ones to acquire space. The pseudo-code of the `CacheUpdate` algorithm (cf. Algorithm 1) is explained below.

**Algorithm 1.** `CacheUpdate`($\mathcal{X}^{\mathcal{B}}$)

```
   Input : $\mathcal{X}^{\mathcal{B}}$
1  for $j = 0$ to $|\mathcal{X}^{\mathcal{B}}|$ and $T_j$ has a cache do
2      foreach referenced tuple $x \in T_j$ do  $Q_{i,j}(x)$=(1+$\epsilon$)*H($x$) + M($x$)
3  for $j = 0$ to $|\mathcal{X}^{\mathcal{B}}|$ and $T_j$ has a cache do
4      foreach referenced tuple $x \in T_j$ do
5          if $j > 0$ then                                          /* not target */
6              $Q_{i,j}(x)$+= H($x$) + (1+e)*M($x$)
7              $Q_{i,j}(x)$+= PropInfo($x, T_{j+1}, t_i$)
8          $G(x) = Q_{i,j}(x)$
9          foreach $u = i - L$ to $i - 1$ do  $G(x) += Q_{u,j}(x) * decay(t_u, t_i)$
10     sort $G$ [desc]
11     $C_{i,j} \leftarrow$ top-$|C_j|$ tuples of $G$
```

The algorithm `CacheUpdate` is invoked *in turn* for each stream $T_j$ that has a cache associated with it (line 1). For each tuple $x \in T_j$ referenced during $t_i$, we count the number of times it was referenced by tuples from another stream $T_u$ that $T_j \xrightarrow{1-to-M} T_u$. We denote as $H(x)$ the hits, i.e. the number of times $x$ was referenced and found in the cache $C_{i,j}$, and as $M(x)$ the misses, i.e. the number of times $x$ was referenced and not found in $C_{i,j}$; obviously, only one of the two values is non-zero at any timepoint. As can be seen in line 3, the importance of tuple $x$ at $t_i$ is computed as the number of references to it, thereby assigning a higher weight $(1 + e)$ to hits, i.e. to tuples already in the cache.

The cache of the target stream is treated separately (lines 6 and from 9 on): the *gain* to be achieved by caching tuple $x \in T_0$ is computed by considering the value computed at line 3 and the number of references to $x$ within the whole sliding window of length $L$ (line 11). To reduce the influence of old statistics, we use the decay function depicted in Eq. 1: it lowers the weight of old data exponentially, unless there is a known periodicity with period $P > 0$. If the data are periodic, tuples that correspond to the same period as the current one modulo P are rejuvenated by assinging to them a weight $p \in (0, 1]$. The referenced tuples are sorted (line 12) and the top-$|C_0|$ positions are kept in the cache (line 13).

$$decay(t_u, t_i) = \begin{cases} 1 & t_i - t_u = 0 \\ p & t_i > t_u \wedge P > 0 \wedge (t_i - t_u) mod P = 0 \\ e^{t_u - t_i} & otherwise \end{cases} \tag{1}$$

The gain computation $G()$ (line 11) aims to maximize the number of tuples contributing to the model. It thus favors tuples that are expected to produce more output when they are joined with inter-linked streams. This corresponds to the MAX-Subset error measure proposed in [12] for sliding-window joins.

Cache updating is an iterative process depicted in lines 4 to 13: a cache is updated by removing tuples from it and inserting new ones; this affects the hits and misses for tuples referenced by them. To explain the effects and side-effects of cache updating, we first consider three streams three streams $T_1, T_2, T_3$ from $\mathcal{X}^{\mathcal{B}}$ that use caches and constitute a chain of relationships $T_1 \xrightarrow{M-to-1} T_2$ and $T_2 \xrightarrow{M-to-1} T_3$. At $t_i$, let $x \rightarrow y$, where $x \in T_1, y \in T_2$. Assume that during cache updating at $t_i$, tuple $y$ is fetched and cached in $C_{i,2}$. If $y \rightarrow z$ for some $z \in T_3$, then $y$ contains a possibly dangling reference; the need to fetch $z$ was not known before $y$ was cached. Indeed, $z$ can only be fetched to the cache of $T_3$ only at the next timepoint $t_{i+1}$. In the general case of an arbitrary length *chain* of M-to-1 relations $T_{j_1}, T_{j_2}, \ldots, T_{j_k}$, it will take $k-1$ timepoints to amend a cache miss for tuple $x \in T_{j_1}$ that propagates to the last stream in the chain. In other words, only after $k-1$ timepoints will it be possible to reconstruct tuple $x$, assuming (optimistically) that all tuples needed are still cached at $t_{i+k-1}$.

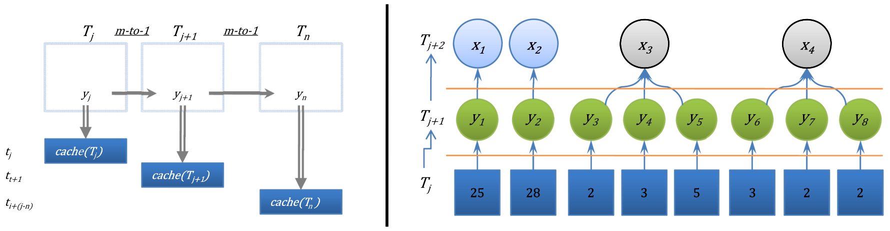


**Fig. 2.** (a) A chain of relationships and (b)transitive references

To load all tuples with their references, we update the caches iteratively, starting with the cache of the target stream (if any). Once the first/next cache has been updated, we calculate the references to the tuples of the remaining streams anew (line 7). This time we assign a higher weight $(1 + e)$ to the misses, making referenced tuples outside the cache more likely candidates for caching.

Line 8 of the `CacheUpdate` algorithm deals with a problem of transitivity across a chain of streams. Consider the example in Fig. 2(b), where all streams are associated with caches and each stream is in M-to-1 relation with the next one. Assume that tuples $x_3, x_4 \in T_j$ are referenced more from $T_{j-1}$ than any other tuple, so they would be preferred over tuples $x_1, x_2$. This is consistent with the MAX-subset error measure that promotes tuples resulting in larger output. Now consider the 3-way join $T_j \bowtie T_{j-1} \bowtie T_{j-2}$. Tuples $x_3$ and $x_4$ would be again preferred over $x_1$ and $x_2$, and this will be inconsistent under the MAX-subset error measure [12], because $x_1, x_2$ are going to produce more output tuples (cf. Fig. 2). The reason for the inconsistency are the tuples referenced by tuples referenced by $x_1, \ldots, x_4$. Tuple referencing is transitive; we exploit this property in that we add transitive references to the hits, resp, misses of a tuple $x$ (line 8). The responsible function $PropInfo()$ is depicted in Eq. 2.

$$PropInfo(x, T_u, t_i) = \sum_{y \in T_u \wedge x \to y} \frac{Q_{u+1,i}(y) + PropInfo(y, T_{u+1}, t_i)}{D} \qquad (2)$$

This function takes as input a tuple $x$ and a stream $T_u$ from a chain $\mathcal{T}$ of streams, such that $T_u \xrightarrow{M-to-1} T_{u+1}$. It computes the references of $x$ to $T_u$ and invokes itself again for each tuple $y$ referenced by $x$. At each invocation it uses an *information decay* parameter $D$. This parameter ensures that the impact of those additionally counted references drops as we navigate down the chain of interlinked streams. The value of gain (line 9), as adjusted with $PropInfo()$ is consistent with the MAX-subset measure [12].

## 3.2 Transforming the Multi-table Streams into one Stream

After cache updating, we map the data in the caches and windows of the multi-table streams into a single stream. For this, we extend the propositionalization algorithm `RelAggs` for static data [6]. The core idea of propositionalization is:

> Each tuple $x$ of the target table $T_0$ is expanded by joining it with all tuples that refer to it (via external key id). If $x$ joins with more than one tuples $x_1, \ldots, x_m$, then these tuples must be summarized into a single row. This is done by adding columns to the schema of $T_0$ for each distinct attribute value $v$ that appears in the set $\{x_1, \ldots, x_m\} =: \mathit{matches}(x)$. This column contains the number of times $v$ appears in $\mathit{matches}(x)$.

We describe how propositionalization is performed in a stream setting by joining and aggregating the tuples in the maintained windows and caches.

**Propagating the Identifiers of the Target Stream.** At each timepoint $t_i$, let $Y_{i,0}$ be the set of tuples in the cache or window associated with the target stream $T_0$. Further, let $T_j$ be a stream that directly references $T_0$ and let $Y_{i,j}$ be the contents of the cache or window associated with it. The key identifiers of the $T_0$ tuples in $C_{i,0}$ are propagated to $Y_{i,j}$.This corresponds obviously to a semijoin upon the stream contents seen thus far. This task is repeated in depth-first manner for each $T_j$: the identifiers of $C_{i,0}$ are propagated and incorporated to the tuples of each stream that references the identifiers of $T_j$. The algorithm for this task is depicted in the Algorithm 2 below: it takes as input the schema $\mathcal{X}^{\mathcal{B}}$ which encompasses the tables and the foreign key relationships among them, transforms it into a tree that has the target $T_0$ as root and the streams referencing it as children. The tree edges are the foreign key relationships, across which the identifiers of $T_0$ are propagated towards the leaf nodes. The propagation is performed with the insertion and query operation on lines 5 and 6 respectively.

**Algorithm 2.** `PropagateIds`($\mathcal{X}^{\mathcal{B}}$)

```
Input : $\mathcal{X}^{\mathcal{B}}$ in tree form
1 foreach $T$ and $T'$ do      /* traversed in depth-first order, first $T$ is $T_0$ with $T'$ */
                                /* as its leftmost child; $Y, Y'$ are their caches/windows */
2     if $T'$ doesn't contain targetId then            /* $T'$ is child and $T$ is parent */
3         INSERT INTO Star($Y'$)
4         SELECT $Y$.targetId, $Y'$.* FROM $Y, Y'$ WHERE $Y'$.joinId=$Y$.id
```

In line 4 of the `PropagateIds` algorithm we see the term "Star". The term is used in `RelAggs` [6] in the context of transforming the original schema $\mathcal{X}^{\mathcal{B}}$ into a star $\mathcal{X}^{\mathcal{S}}$ that has the target table $T_0$ as its center. In the context of data warehousing, this operation roughly corresponds to transforming a snowflake schema into a single table. In Fig. 3 (left side) we see the data of the original schema in Fig. 1 and the star schema with the target `Customer` at the center.

**Propositionalization of the Target Stream.** At timepoint $t_i$, the current contents of $T_0$ in the cache or window $Y_{i,0}$ must be "propositionalized" to accommodate the information of each cache or window $Y_{i,j}$ of stream $T_j$. We first consider propositionalization for $t_i, i = 1$: we perform the semijoin between the cache or window $Y_{i,0}$ of the target table $T_0$ and the cache or window $Y_{i,j}$ of each stream $T_j$ that is in 1-to-1 or 1-to-M relationship with $T_0$. Accordingly, we expand $x \in Y_{i,0}$ with the matching tuple $y \in Y_{i,j}$, i.e./ $x := x + +y$. At a later timepoint $t_i, i > 1$, if $x$ is already expanded with some (old) tuple of $Y_{i,j}$, this old expanded tuple is replaced by the new one. This initially unintuitive approach is best explained by an example: assume the stream of students enrolled in a faculty (target stream) and the stream of examination results, as processed by the examination office; a student has exactly one note value per course (including the value NULL), so once a new value comes, it must replace the previous one.

For each $T_j$ that is in 1-to-M or M-to-N relationship to $T_0$, we associate each tuple $x \in Y_{i,0}$ with the set of matching tuples $matches(x) \subseteq Y_{i,j}$. The tuples in this set must be summarized in a single *subtuple*. We distinguish two cases: $x$ is a *compute* tuple, i.e. a tuple that is either new or was inactive at the previous timepoint $t_{i-1}$, or it is a *re-compute* tuple, i.e. was already active in $t_{i-1}$. The reason for this distinction is that tuples seen at $t_{i-1}$ have been already considered for the summarization. Hence, *compute* tuples are joined only to new ones, while *re-compute* tuples are joined to all tuples in the window or cache.

For the summarization of the values of each numerical attribute $A$ among the $matches(x)$, we generate four attributes and accommodate in them the min, max, count and average of the $A$ values seen in $matches(x)$. In the example on Fig. 1(a), the attribute `Price` is numeric and customer 1"David" has

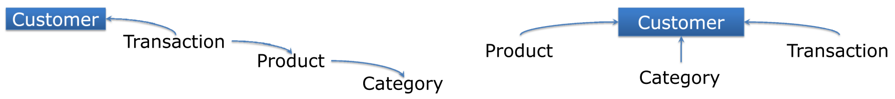


**Fig. 3.** Original schema (left) and star schema (right) for `RelAggs` [6]

**Algorithm 3.** Multi-Table Stream Propositionalization($\mathcal{X}^{\mathcal{B}}, \mathcal{X}^{\mathcal{C}}, T_0$)

```
Output: Single-Stream
1  PropagateIds($\mathcal{X}^{\mathcal{B}}$)
2  $list$ = GetStreams($\mathcal{X}^{\mathcal{S}}$)
3  for $j \leftarrow 1$ to $|list|$ do
4      if $T_0 \xrightarrow{1-1} T_j$ OR $T_0 \xrightarrow{m-1} T_j$ then $\oplus$ tuples of $Y_j$ with $Y_0$
5      else if $T_0 \xrightarrow{1-m} T_j$ then
6          GROUP BY operation on $T_0.id$
7          foreach attribute $A \in \mathcal{S}_j$ do
8              if IsNum($A$) then $\mathcal{S}_{i,0} \leftarrow \oplus\,(avg(A), sum(A), min(A), max(A))$
9              if IsNom($A$) then $\mathcal{S}_{i,0} \leftarrow \oplus_{c \in columns(A,t_i)} \left( \left| \sigma_{v \in values(c,t_i)}(A) \right| \right)$
10 CacheMaintenanceProcedure($\mathcal{X}^{\mathcal{B}}$)
11 AdjustNominal($\mathcal{X}^{\mathcal{B}}$)
```

purchased two products; the prices of these products are summarized in the columns MIN_P,MAX_P,AVG_P of Fig. 1(c) while COUNT_P accommodates the number of purchases - two. Differently from the students' example above, the four generated attributes are *updated* for each tuple $x$, not replaced.

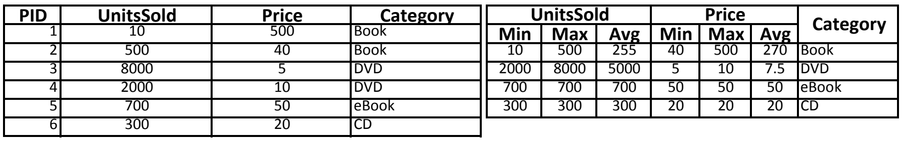
For summarization of each nominal attribute, function AdjustNominal($\mathcal{X}^{\mathcal{B}}$) (cf. Line 11, Algorithm 3) generates as many columns as there are distinct values in $matches(x)$. In the static scenario, this is feasible because we know the number of distinct values for $A$ in advance. In the stream scenario, the schema can neither be expanded arbitrarily (memory is finite) nor can each value ever seen be retained perpetually in it (memory must be used efficiently). We rather specify for each nominal attribute $A$ in the original schema $\mathcal{X}^{\mathcal{B}}$ an upper boundary $r_A$ to the number of columns/positions that can be reserved for it in $Y_{i,0}$ at any $t_i$. The space allocated initially for $A$ is possibly less; it is the maximum of $domain(A, 1)$ - the number of distinct values seen for $A$ at the first timepoint - and a user-defined lower boundary $l_A$ (cf. Table 1). In the example of Fig. 1, the attribute Category for products takes the values "Book" and "DVD". If $l_A = 2$ as well, then $columns("Category", 1) = 2$.

At $t_i, i > 1$ the number of values for $A$ to be accommodated may increase, e.g. if the company decides to introduce the categories "eBook" and "CD". Hence, we must adjust the available space to the demand. For this, we propose a *soft adjustment* and a *hard adjustment* method. Briefly, soft adjustment implies deleting some of the past values to acquire space for the new ones, while hard adjustment encompasses the grouping of values into $r_A$ groups. We explain these space adjustment methods below.

**Space Adjustment.** At some timepoint $t_i$, let $A$ be a nominal attribute of some stream $T_j$ and let $domain(A, i)$ be the set of values that $A$ acquires at $t_i$. These are the values to be accommodated. The cache or window $Y_{i,0}$ of $T_0$ contains columns for $domain(A, i-1)$, i.e. for the values of $A$ accommodated at the previous timepoint. The number of currently reserved columns $columns(A, i-1)$ is not necessarily equal to $|domain(A, i-1)|$; it may be larger but is certainly not exceeding the number of positions $r_A$ reserved for $A$.

If $|domain(A, i) \setminus domain(A, i-1)| \leq r_A$, then the new values are accommodated in some of the remaining reserved positions and the tuples $Y_{i,0}$ are expanded. Otherwise, some positions must be freed to accommodate the values in $domain(A, i)$. If $|domain(A, i))| \leq r_A$, then some of the old values (from $domain(A, i-1)$) are replaced. This means that those columns in $Y_{i,0}$ that have been generated by propositionalization of $T_j$ acquire new semantics. This case is best perceived if one observes customers (target stream) and the products they purchase: the longer one customer interacts with the shop, the more products s/he has purchased. However, for the learning of a model, it is reasonable to forget some of the products purchased long ago for the sake of allowing new products to contribute more to the model. Our *soft adjustment* heuristic de-allocates $|domain(A, i) \setminus domain(A, i-1)|$ columns among those accommodating values from $domain(A, i-1) \setminus domain(A, i)$. It then assigns the free positions to these new values. Currently, the columns to be de-allocated are selected at random, but more sophisticated options are possible, e.g. eliminating the oldest values.

If $|domain(A, i)| > r_A$, then we *cluster* the values in $domain(A, i)$ into $r_A$ clusters. Clustering implies that values appearing in the same cluster are accommodated in one column and become indistinguishable. In our example, let $r_{Categry} = 2$, $values(Category, i) = \{$`eBook,CD,DVD`$\}$ and $columns(Category, i) = \{$`Book,DVD`$\}$. Then the set $\{$`Book,eBook,CD,DVD`$\}$ must be stored in two columns. Hence, this *hard adjustment* method incurs information loss.



| PID | UnitsSold | Price | Category |
|---|---|---|---|
| 1 | 10 | 500 | Book |
| 2 | 500 | 40 | Book |
| 3 | 8000 | 5 | DVD |
| 4 | 2000 | 10 | DVD |
| 5 | 700 | 50 | eBook |
| 6 | 300 | 20 | CD |

| UnitsSold | | | Price | | | Category |
|---|---|---|---|---|---|---|
| Min | Max | Avg | Min | Max | Avg | |
| 10 | 500 | 255 | 40 | 500 | 270 | Book |
| 2000 | 8000 | 5000 | 5 | 10 | 7.5 | DVD |
| 700 | 700 | 700 | 50 | 50 | 50 | eBook |
| 300 | 300 | 300 | 20 | 20 | 20 | CD |

**Fig. 4.** (a)Dependant nominal values converted to (b)feature vector for clustering

Clustering is based on semantic similarity among the objects. Here, the objects are distinct values of an attribute. We assert that *two values are similar if the tuples containing them are similar.* This implies modeling a value as the set or table of all tuples containing it. As we have already pointed out in Section 1, mining algorithms assume tuples to be independent. Thus, similarly to the propositionalization solution of summarizing tuples into a single one, we summarize the tuples that have same value into one large tuple by turning rows in columns. The generated feature vectors (cf. Fig. 4b) are independent and we use Hierarchical Flat Clustering [19] with $k$=$r_A$ to cluster them.

## 4 Experiments

We have evaluated our method on the "Gazelle dataset" of the KDD Cup 2000 [4][20] and on the "Financial dataset" of the PKDD Challenge 1999 [5]. Our

[4] http://cobweb.ecn.purdue.edu/KDDCUP/data/

[5] http://lisp.vse.cz/challenge/

objective was to study the performance of our multi-table stream propositionalization and mining approach. To this purpose, we have designed a reference for multi-table incremental clustering, in which we assume infinite data storage.

## 4.1 The Datasets

The Gazelle dataset consists of several tables that describe the activities of users in an web commerce site in the period 02-05/2000. The tables in the Financial dataset depict the activities of bank customers who have been granted a loan and are paying it back in the period 01/1993-12/1998. The Financial dataset has labeled data, so we have tested our strategies against the ground truth.

**Table 2.** Tables and statistics from (a) the Gazelle and (b) the Financial datasets

| *Gazelle dataset* | | | *Financial dataset* | | | | |
|---|---|---|---|---|---|---|---|
| | Table | Tuples | | Table | Tuples | (A/C) | (B/D) |
| | `Customers` | 3336 | $C$ | <u>`Accounts`</u> | 682 | 606 | 76 |
| $W$ | <u>`Sessions`</u> | 18113 | $C$ | `Districts` | 77 | | |
| $W$ | `Request` | 213,101 | | `Clients` | 827 | | |
| $C$ | `Products` | 376 | $W$ | `Orders` | 1513 | | |
| $C$ | `Contents` | 77 | | `Cards` | 170 | | |
| | | | $W$ | `Transactions` | 191,556 | | |

In Table 2 we depict the tables of each dataset; the target table is underlined. In the first column, we mark with $C$ each stream associated with a cache, while $W$ stands for window. In the Gazelle dataset, we build user profiles on products inspected by each user (user "request") during each session, so sessions without product requests are dropped. There has been no information about re-visits of users, so a user corresponds to a session, and `Sessions` is the target. The contents of a session may be of several types, depicted in the table `Content`.

In the Financial dataset, a loan is associated with an account, which in turn may belong to one or more clients. Credit card activities and orders are recorded, but the main load comes from the `Transactions` stream. Originally, the dataset had four classes. Already during the 1999 competition, the classes A, C and B, D were merged into *loan-trusted* and *loan-risk* respectively. We do the same.

## 4.2 Caching Strategies and Reference Strategy

Our hypothesis is that the amount of information remembered as the multi-table stream progresses has an impact upon the quality of the clustering results. The remembered information is affected by the cache size and by the number of columns reserved for each nominal attribute $A$. We have thus varied these values for the tables `Products, Contents` of the Gazelle dataset and for `Accounts, District` of the Financial dataset. To control the impact of the reserved columns better, we have set $l_A = r_A =: r$ for all attributes. The specification of cache size and reserved columns is a "cache strategy". The strategies we used are in Table 3. For example, strategy "FIN2" uses a cache of 200 account' and 40 district's tuples and reserves 3 columns for storing nominal values of each attribute.

**Table 3.** Strategies on the Gazelle and the Financial datasets

| *Gazelle Dataset* | | | | | | *Financial Dataset* | | | |
|---|---|---|---|---|---|---|---|---|---|
| *Acronym* | G1 | G2 | G3 | G4 | G5 | *Acronym* | FIN1 | FIN2 | FIN3 |
| `Products` | 5 | 25 | 75 | 150 | 200 | `Accounts` | 100 | 200 | 300 |
| `Contents` | 5 | 15 | 40 | 60 | 70 | `District` | 20 | 40 | 50 |
| $r$ | 7 | 7 | 7 | 7 | 14 | $r$ | 3 | 3 | 3 |

Our reference strategy has unlimited storage and knows the future. It thus does not need cache maintenance, nor information propagation. Also, $r$ is large enough to accommodate all nominal values that will come in the future.

The window size is the same for all strategies, including the reference. For the Financial dataset we used a window $w$ of 30 months, for the short-lived data of the Gazelle dataset we experimented with $w = 14, 21$ where each window contains $w \times 75$ propositionalized sessions. For clustering we used Online-K-Means [11] with cosine as measure of tuple similarity. For Gazelle, the number of clusters was set to $K = 5$, while Financial dataset uses $K = 3, 9$.

## 4.3 Evaluation Measures

At timepoint $t_i$, the incremental clustering algorithm adjusts the clusters of model $\zeta_{i-1}$ into model $\zeta_i$ [6]. If the quality of $\zeta_i$ is lower than that of $\zeta_{i-1}$ below a threshold $\tau = 0.7$, then the data are re-clustered.

We use following quality measures from [21]: *silhouette coefficient* to measure the quality of one model/clustering; *Jaccard coefficient* to compare clusterings at different timepoints and for different strategies; *entropy* to evaluate a clustering against explicit class labels (only available for the Financial dataset).

To compute the silhouette coefficient of a tuple $x$ in cluster $C \in \zeta$, we calculate its average distance $a_x$ from all other tuples in $C$ and from the tuples in the clusters of $\zeta \setminus \{C\}$, say $b_x$. Then $s(x) = \frac{(b_x - a_x)}{\max(a_x, b_x)}$. The silhouette of $C$ is the average silhouette of its members. The silhouette for the clustering $silhouette(\zeta)$ is the average over the cluster silhouettes, weighted with cluster cardinalities.

For the Jaccard assume two clusterings $\zeta, \zeta'$. Let $f11$ be the number of records for which $\zeta, \zeta'$ agree that they should be in the same cluster. Let $f10$ be the number of records that were put in the same cluster by $\zeta$ but in distinct clusters by $\zeta'$ (disagreement), and let $f01$ be the number of records put in the same

[6] A model built by a clustering algorithm is a set of clusters - a "clustering". We use the nouns "model" and "clustering" as synonyms hereafter.

cluster by $\zeta'$ but in distinct clusters by $\zeta$ (disagreement). Then, the Jaccard coefficient is $JaccardCoeff(\zeta,\zeta') = \frac{f11}{f11+f01+f10}$ [21].

The entropy measures the degree to which a cluster contains tuples belonging to a single class. Let $\zeta$ be a clustering and $\xi$ be the set of classes describing the data. For each $C_u \in \zeta$ and $L_v \in \xi$, the probability that a tuple in $C$ belongs to $L_v$ is $p_{uv} = \frac{|C_u \cap L_v|}{|C_u|}$. The entropy of $C_u$ is $e(C_u,\xi) = \sum_{L_v \in \xi} p_{uv} log_2 p_{uv}$. The entropy of $\zeta$ is then $entropy(\zeta,\xi) = \frac{\sum_{C_u \in \zeta} |C_u| e(C_u)}{|\cup_{C_v \in \zeta} C_v|}$; *lower* values are better.

## 4.4 Findings with the Financial Dataset

In the Financial dataset, we use a cache for the stream `Accounts`; at each timepoint it accommodates a set of accounts and, after propositionalization, of the transactions on them. Each account arrives with zero transactions and *evolves* into either "loan-trusted" or "loan-risk" class as more and more transactions are recorded for it. To avoid learning our models on data that arrive early but are not relevant (and would thus blur our results in an undisciplined way), we have trained a classifier (J4.8 [22]) on it, identified the subset of predictive attributes and then projected the remaining attributes away to reduce the noise. We varied the number of columns reserved for each nominal attribute: $r = 3$ and $r = 7$, but we found that the results were almost identical, except for some slight variations between timepoints $t_{30}$ and $t_{40}$. Therefore, we report the results of $r = 3$ only.

In the left side of Fig. 5, drawn for $K = 9$ clusters, we show the entropy values when comparing each cache strategy against the ground truth. At the beginning, all strategies have an entropy value of zero, because the first accounts belong all to the same class. As soon as accounts from the other class arrive, the entropy rises sharply.

It must be stressed here that accounts are dynamic objects that mature and evolve over time. When accounts are introduced, the clusterer is only aware of their initial static properties e.g. the information about the owner(s) and types of card they hold, the district they were created in and etc. There is little or no transaction information associated with them, so they are initially grouped into clusters on the basis of these static properties. On the other hand, the

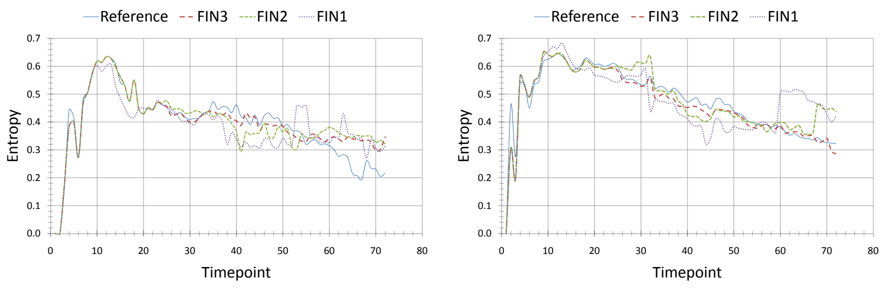


**Fig. 5.** Alternate cache strategies for $w = 30$ (*Financial*) (a)$K = 9$ and (b) $K = 3$

class label reflects their *final* state, after many transactions have accumulated on them. This is the reason that all strategies perform poorly at the beginning.

Around timepoint $t_{10}$, the entropy of FIN1 (the strategy with the smallest cache) starts dropping. By this time, more than 100 accounts have arrived, and the `CacheUpdate` algorithm (Section 3.1) prefers those that have performed more transactions. For FIN1 with its small cache, this means that accounts with few transactions are not in the cache. All other strategies, including the reference strategy, have larger caches and store these accounts, which cannot be easily classified as loan-risk vs loan-trusted, and thus result in bad performance. At later timepoints, i.e. after $t_{23}$ for FIN2 and after $t_{30}$ for FIN3, these strategies also drift away from the reference strategy: as they reach their cache size limit, they keep only mature accounts inside the cache, so their performance increases.

From timepoint $t_{32}$ until $t_{55}$, the reference strategy with its infinite cache shows the worst performance. The lesson learned is that in stream mining it is not always desirable to remember all the data. For the Financial dataset, oblivion is best: FIN1 that has the smallest cache size outperforms all other strategies.

After timepoint $t_{55}$, only very few new accounts arrive, the last one at $t_{60}$. For the next 12 timepoints, all accounts keep evolving as new transactions arrive for them. The performance of the reference strategy also improves towards the end since there are no noisy accounts to perturb it. So it outperforms all strategies as it there is no information loss due to memory limitations.

In the right side of Fig. 5, we show the cache strategies for $K = 3$ clusters. We chose a small $K$ to test whether the tuples of the majority class A/C are better accommodated in few large clusters. $K = 3$ turned out to be not a goof choice for the data with a very skewed distribution of classes and very little separation between the data points. Before $t_{32}$ when information about accounts is very little, all the strategies perform very poorly. Its performance starts getting better as information pours in. However, it should be noted that all the strategies were able to separate a large number *loan-risk* accounts into one cluster but the cluster also contained almost the same amount (or even more at some timepoints) of *loan-trusted* accounts while the other two clusters were mostly clean. This explains the relatively low entropy for the strategies with $K = 3$. Whereas several small clusters with $K = 9$ were able achieve separation by discovering clusters that contained only the *loan-risk* accounts.

## 4.5 Findings on the Gazelle Dataset

The Gazelle dataset has no ground truth, so we study the performance of our strategies towards the reference. We also mark the timepoints of re-clustering.

In the left side of Fig. 6 we see the behavior of each strategy for a sliding window of 14 units, where a unit is 75 propositionalized sessions. All strategies are initially comparable to the reference, but their performance deteriorates as records are forgotten. The large-cache strategies G4 and G5 have inferior performance than small-cache strategies. This indicates a tendency for new products and contents, in which small-cache strategies adapt fast.

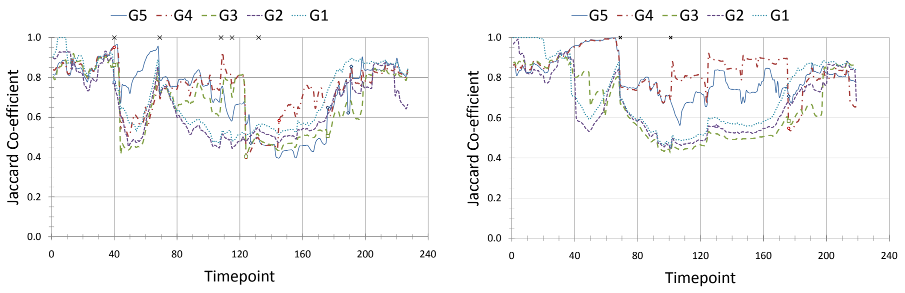


**Fig. 6.** Alternate cache strategies (*Gazelle*) (a)$w = 14$ and (b)$w = 21$

A concept shift occurs soon after timepoint 40, whereupon the reference and some of the cache strategies experience re-clustering. The timepoint fits with a TV advertisement for Gazelle, which is known to have led to an increase in the site traffic. All strategies are slow in adapting to this concept shift, but the large-cache strategy G5 is the first to show an upward trend. This indicates that the range of products flowing into user profiles by that time is large.

The next concept shift occurs just before timepoint 70. After this shift, the large-cache strategies perform closer to the reference, while small-cache strategies do not perform well and re-cluster around timepoint 125. Two more shifts cause re-clustering of the reference a little earlier; they were both accompanied by a performance degradation for the large-cache strategies. It is remarkable that the strategy G4 has been close to the reference, although it did not re-cluster itself. This indicates that the clusters under this strategy were adapted adequately to the changes in the population. This also holds at the last shift around timepoint 135: the performance of the large-cache strategies deteriorates, but G4 experiences re-clustering only once. This indicates that a smaller cache leads to a better adaptation during that period. This claim is further supported by the small-cache strategies, whose performance improves without re-clustering. We interpret this as a tendency of the users to concentrate on few products.

On the right side of Fig. 6, we see the behavior of the strategies for a window size $w = 21$. We can observe the same phenomena, i.e. shifts and re-clustering, although the first shift is captured much later than for $w = 14$. Small-cache strategies show a steeper performance deterioration in early timepoints but improve in late timepoints and become more competitive. Until timepoint 175, G4 performs better than G5, which uses an even larger cache. At that timepoint, G4 shows very poor performance but improves a bit after re-clustering, although its quality is inferior to that of the small-cache strategies.

These results indicate that multi-table stream clustering allows for adaptation to shifts. Small-cache strategies adapt without need of quality monitoring and perform better when the data are very volatile. Large-cache strategies have higher performance when there is some stability in the data, but require quality monitoring to respond to concept shifts. As for many phenomena on streams, the selection of a proper window size is of crucial importance. However, cache strategies allow for the incorporation of influence from data that do not belong to the target table, while softening the impact of the window size selection. As we have seen, even with a larger window size, caching allows for the adaptation to shifts, although these shifts are recognized with some delay.

## 5 Conclusion

We studied the new problem of multi-table stream mining and presented an incremental clustering method for it. Our method encompasses a cache-and-window management strategy and a stream transformation algorithm. The former makes data available for model building and adaptation, favouring tuples that produce larger output. The latter transforms the interrelated streams into a single-stream. It thereby joins and summarizes the data into a single table of a fixed schema, while ensuring that even an 1-to-M semjoin results in a *single* output tuple per input tuple of the left join operand.

To study the performance of our approach we have designed a reference strategy that knows the future and has unlimited resources. On this basis we have experimented with two datasets that are of multi-stream nature and extrapolated a ground truth by one of them. We have shown that our approach approximates the reference well and outperforms it in those cases where oblivion is preferable - in response to concept drifts and shifts.

As a next step, we want to study the potential of data sampling and investigate more elaborate caching strategies We further intend to devise a method to accommodate the stream data while minimizing the schema size.

**Acknowledgement.** We thank Blue Martini Software for contributing the KDD Cup 2000 data.

## References

1. Blockeel, H., Raedt, L.D.: Top-down induction of first-order logical decision trees. Artificial Intelligence 101(1-2), 285–297 (1998)
2. Dehaspe, L., Toivonen, H.: Discovery of relational association rules. In: Relational Data Mining, pp. 189–212. Springer, Heidelberg (2001)
3. Dehaspe, L., Toivonen, H.: Discovery of frequent datalog patterns. Data Min. Knowl. Discov. 3(1), 7–36 (1999)
4. Dzeroski, S., Lavrač, N.: Inductive learning in deductive databases. IEEE TKDE 5(6), 939–949 (1993)
5. Kramer, S., Widmer, G.: Inducing classification and regression trees in first order logic. In: Relational Data Mining, pp. 140–156. Springer, Heidelberg (2001)
6. Kroegel, M.A.: On Propositionalization for Knowledge Discovery in Relational Databases. PhD thesis, Otto-von-Guericke-University Magdeburg, Germany (2003)
7. Lavrač, N., Flach, P.: An extended transformation approach to inductive logic programming. ACM Trans. Comput. Logic 2(4), 458–494 (2001)

8. Bartolini, I., Ciaccia, P., Ntoutsi, I., Patella, M., Theodoridis, Y.: A unified and flexible framework for comparing simple and complex patterns. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) PKDD 2004. LNCS, vol. 3202, pp. 496–499. Springer, Heidelberg (2004)
9. Maddalena, A., Catania, B.: Towards an interoperable solution for pattern management. In: 3rd Int. Workshop on Database Interoperability INTERDB 2007 (in conjunction with VLDB 2007), Vienna, Austria (September 2007)
10. Guha, S., Meyerson, A., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams: Theory and practice. IEEE TKDE 15(3), 515–528 (2003)
11. Beringer, J., Huellermeier, E.: Online clustering of parallel data streams. Data & Knowledge Engineering 58(2), 180–204 (2006)
12. Das, A., Gehrke, J., Riedewald, M.: Approximate join processing over data streams. In: Proc. of SIGMOD 2003 (2003)
13. Srivastava, U., Widom, J.: Memory-limited execution of windowed stream joins. In: Proc. of VLDB 2004, VLDB Endowment, pp. 324–335 (2004)
14. Xie, J., Yang, J., Chen, Y.: On joining and caching stochastic streams. In: Proc. of SIGMOD 2005, pp. 359–370. ACM, New York (2005)
15. Muggleton, S., Raedt, L.D.: Inductive logic programming: Theory and methods. J. Log. Program. 19/20, 629–679 (1994)
16. Emde, W., Wettschereck, D.: Relational instance based learning. In: Saitta, L. (ed.) Proc. of ICML 1996, pp. 122–130. Morgan Kaufmann, San Francisco (1996)
17. Kirsten, M., Wrobel, S., Horváth, T.: Distance based approaches to relational learning and clustering. In: Rel. Data Mining, pp. 213–230. Springer, Heidelberg (2001)
18. Knobbe, A.J., de Haas, M., Siebes, A.: Propositionalisation and aggregates. In: Siebes, A., De Raedt, L. (eds.) PKDD 2001. LNCS, vol. 2168, pp. 277–388. Springer, Heidelberg (2001)
19. Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., Euler, T.: Yale: Rapid prototyping for complex data mining tasks. In: Ungar, L., Craven, M., Gunopulos, D., Eliassi-Rad, T. (eds.) Proc. of KDD 2006, pp. 935–940. ACM, New York (2006)
20. Kohavi, R., Brodley, C., Frasca, B., Mason, L., Zheng, Z.: KDD-Cup 2000 organizers' report: Peeling the onion. SIGKDD Explorations 2(2), 86–98 (2000)
21. Tan, P.N., Steinbach, M., Kumar, V.: Intro. to Data Mining. Wiley, Chichester (2004)
22. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)

# Improving Relation Extraction by Exploiting Properties of the Target Relation

Eric Normand, Kevin Grant, Elias Ioup, and John Sample

Naval Research Laboratory
Stennis Space Center, MS
{enormand,kevin.grant.ctr,eioup,john.sample}@nrlssc.navy.mil

**Abstract.** In this paper we demonstrate and quantify the advantage gained by allowing relation extraction algorithms to make use of information about the cardinality of the target relation. The two algorithms presented herein differ only in their assumption about the nature of the target relation (one-to-many or many-to-many). The algorithms are tested on the same relation to show the degree of advantage gained by their differing assumptions. Comparison of the performance of the two algorithms on a one-to-many domain demonstrates the existence of several, previously undocumented behaviors which can be used to improve the performance of relation extraction algorithms. The first is a distinct, inverted u-shape in the initial portion of the recall curve of the many-to-many algorithm. The second is that, as the number of seeds increases, the rate of improvement of the two algorithms descreases to approach the rate at which new information is added via the seeds.

## 1 Introduction

The amount of information available on the Internet as unstructured text has made text mining an increasingly important area of study. One aspect of text mining is *relation extraction*. Relation extraction is the process by which the relationship between entities in a text is identified and characterized. The result of a relation extraction process is typically a table relating pairs of entities. Often, properties of the relation being extracted, such as uniqueness constraints on one element, can be exploited in the extraction algorithm. In this paper, we conduct an experiment to measure the benefit of allowing an algorithm to make use of such properties.

Each of the two algorithms presented is specialized to one common type of binary relation. The first is specialized to many-to-many relations, and the second to one-to-many relations. A one-to-many relation is a relation such that each value for the first attribute of a tuple may appear in many tuples while the second value may appear only once (it is unique). For example one mother can have many children, while each child can have only one mother. A many-to-many relation, which is the "null" assumption, since it places no constraints on the values, is similar to a one-to-many relation but with no uniqueness constraint on the values for the attributes. For example one child 'x' can have many cousins, each of whom can have many cousins other than ‘x’. In this paper we quantify the amount of advantage to be gained by making use of knowledge of the nature of the relation in the relation extraction process.

Section 2 of this paper compares the current work with prior works in the field. Section 3 formally defines the problem and derives the Naive Bayes Classifier used in both algorithms. Section 4 presents the two algorithms in detail. Section 5 describes the experiment. Section 6 discusses the results of the experiment. Section 7 presents our conclusions.

## 2 Prior Works

The extraction algorithms developed for the experiment use a bootstrapping process similar to [1] and [2]. [1] makes no assumptions about the target relation, so implicitly assumes it is many-to-many. [2] requires the relation be one-to-many and makes several improvements to [1] because of the improved specificity of the algorithm. An experiment similar to the current work could be used to measure the degree to which the stronger assumption of [2] gave it an advantage over [1], and how much was due to improvements in the algorithm.

[3] hypothesizes that the most important syntactic and lexical features of a relation occur along the shortest path between the two entities on the dependency graph. We have used this hypothesis in building the algorithms presented herein. However, testing this hypothesis is beyond the scope of this paper.

The algorithms presented herein, like [4], [2], [5], [6], [7], [8], [9], [10], [11], and [12], use features derived from multiple sentences. Any sentence that contains a seed value will potentially generate a feature. The features are then used to classify the values that they are associated with.

## 3 Background

The two algorithms presented in this paper build approximations to a relation from an initially supplied set of seed tuples. Specifically, we subdivide the set of possible tuples into three subsets. The set $S^+$ contains the positive seeds. Positive seeds are known to be part of the relation. The set $S^-$ contains the negative seeds. Negative seeds are known to not be part of the relation. The set $S^?$ contains the unprocessed tuples. It is the task of the relation extraction algorithm to correctly categorize (into the positive and negative sets) as many tuples from $S^?$ as is possible.

The algorithms identify potential relation instances by matching them against a linguistic template (“pattern”). The patterns are used to capture the syntactic and lexical expression of a semantic relation. For example, the pattern “$X$ is the capital of $Y$” may be used as a pattern for identifying potential instances of the relation “capital of”, where the words associated with the labels $X$ and $Y$ would become the attribute values composing one tuple of the relation. Let us call that tuple $t_i = \langle X, Y \rangle$. Let us call the pattern $q$. We say that $matches(q, t_i)$ if there exists a sentence that matches $q$, and $X$ and $Y$ are the corresponding words taken from that sentence. For example, given the pattern above and a sentence "Berlin is the capital of Germany.", we say $matches(q, \langle \text{Berlin}, \text{Germany} \rangle)$. If we can go further and say that $t_i$ should be assigned to the set of tuples that we know to be instances of the relation $R$, we say $t_i^+$. If $t_i$ were already known to be an instance of the relation, we would say $t_i \in S^+$. There are analogous meanings for $t_i^-$ and $t_i \in S^-$.

Initially, each potential relation instance is assigned to one of $S^+$, $S^-$, or $S^?$. The two algorithms presented in this paper identify potential tuples to be moved from $S^?$ to another set based on an estimation of the probability that it belongs in that set. We calculate $P(t_i^+|Q_i)$, $P(t_i^-|Q_i)$, and $P(t_i^?|Q_i)$ where $Q_i$ is the set of patterns $q$ such that $matches(q, t_i)$. The single tuple with the highest probability ratio is selected and moved into the appropriate set.

We also define various notations to simplify the explanation of the algorithm. A superscript on a $Q_i$ indicates that the patterns in the set only match tuples from certain seed sets. For example, $Q_i^{+?}$ is the set of patterns that match $t_i$ and that only match tuples from $S^+$ and $S^?$. Formally:

$$Q_i^{+?} = \{q | q \in Q_i \wedge \forall t_j \neg (t_j \in S^- \wedge matches(q, t_j))\} \tag{1}$$

$Q_i^{-?}$ is defined similarly.

We wish to derive a formula for estimating $P(t_i^+|Q_i)$, $P(t_i^-|Q_i)$, and $P(t_i^?|Q_i)$. Using Bayes' Theorem, we write:

$$P(t_i^+|Q_i) = \frac{P(Q_i|t_i^+)P(t_i^+)}{P(Q_i)} \tag{2}$$

Next, we assume that we can estimate the prior probability from the known data.[1]

$$P(t_i^+) \approx \frac{|S^+|}{|S^+ \cup S^- \cup S^?|} \tag{3}$$

Then we assume that the matches are probabalistically independent, which lets us write:

$$P(Qi|t_i^+) = \prod_{q_j \in Q_i} P(matches(q_j, t_i)|t_i^+) \tag{4}$$

The probability that a pattern $q_j$ matches a tuple $t_i$ given that $t_i^+$ is:

$$P(matches(q_j, t_i)|t_i^+) = \frac{|S^+ \cap N_j|}{|S^+|} \tag{5}$$

[1] This assumption is very strong, since the known data is small (as small as one seed tuple) and the entire set of possible tuples has over 1000 tuples. However, in practice, the influence of the inaccuracy of this estimate is small enough to allow the algorithm to bear good results, as shown in Section 6.

where $N_j$ is the set of tuples matched by $q_j$. We can see that:

$$P(Q_i|t_i^+) = \prod_{q_j \in Q_i} \frac{|S^+ \cap N_j|}{|S^+|} \tag{6}$$

Using these results, we substitute into Equation 2:

$$P(t_i^+|Q_i) = \frac{|S^+|}{|S^+ \cup S^- \cup S^?|P(Q_i)} \prod_{q_j \in Q_i} \frac{|S^+ \cap N_j|}{|S^+|} \tag{7}$$

Similar formulas can be generated for $P(t_i^-|Q_i)$ and $P(t_i^?|Q_i)$.

Using these equations, inequalities like $P(t_i^+|Q_i) > P(t_i^?|Q_i)$ and ratios like $\frac{P(t_i^+|Q_i)}{P(t_i^?|Q_i)}$ can be solved numerically, as many terms cancel out and the remaining terms are numerically calculable. The numbers generated by this process will be used by the algorithms presented in the next section in order to extract a relation.

## 4 Algorithm Definitions

In this section, we present two algorithms for extracting a relation from a document set. Both algorithms are semisupervised and use a small initial set of user-supplied examples (seeds) to bootstrap the learning process. The algorithms differ in the extent to which they incorporate information about the properties of the target relation. The first algorithm makes no assumptions about the relation. Effectively, it assumes that the target relation is many-to-many. The second operates on the (correct) assumption that the target relation is one-to-many. Both algorithms make use of three sets into which tuples are assigned: a set of unassigned tuples $S^?$, a set of tuples judged to be relation instances $S^+$, and a set of tuples judged not to be relation instances $S^-$.

### 4.1 An Algorithm for Many-to-Many Domains

```
Repeat:
    Let $X_1 = \{t_i | t_i \in S^? \wedge P(t_i^-|Q_i^{-?}) > P(t_i^+|Q_i^{-?}) \wedge P(t_i^-|Q_i^{-?}) > P(t_i^?|Q_i^{-?})\}$
    Let $X_2 = \{t_i | t_i \in X_1 \wedge \forall(t_j \in X_1) \frac{P(t_i^-|Q_i^{-?})}{P(t_i^?|Q_i^{-?})} \geq \frac{P(t_j^-|Q_j^{-?})}{P(t_j^?|Q_j^{-?})}\}$
    If $X_2 \neq \emptyset$ then move $X_2$ from $S^?$ to $S^-$.
    Let $Y_1 = \{t_i | t_i \in S^? \wedge P(t_i^+|Q_i^{+?}) > P(t_i^-|Q_i^{+?}) \wedge P(t_i^+|Q_i^{+?}) > P(t_i^?|Q_i^{+?})\}$
    Let $Y_2 = \{t_i | t_i \in Y_1 \wedge \forall(t_j \in Y_1) \frac{P(t_i^+|Q_i^{+?})}{P(t_i^?|Q_i^{+?})} \geq \frac{P(t_j^+|Q_j^{+?})}{P(t_j^?|Q_j^{+?})}\}$
    If $Y_2 \neq \emptyset$ then move $Y_2$ from $S^?$ to $S^+$.
Until $X_2 = \emptyset \wedge Y_2 = \emptyset$
```

First, this algorithm selects tuples from $S^?$ such that the probability that the tuple should be reassigned to $S^-$ is greater than either the probability that it should be reassigned to $S^+$ or that it should remain in $S^?$. From this set, it selects that tuple which maximizes the ratio of the probability that it should be reassigned to $S^-$ over the probability that it should remain in $S^?$. This tuple is moved from $S^?$ to $S^-$. An analogous process is then carried out for tuples that may be reassigned to $S^+$. This entire process repeats until the most recent iteration has produced no changes. In effect, this algorithm iteratively moves tuples from the set of unassigned tuples into one of the other sets until no more tuples pass the criteria for being moved.

### 4.2 An Algorithm for One-to-Many Relations

This algorithm takes advantage of the fact that the target relation is one-to-many by constructing $S^-$ directly from $S^+$. This can be done because once a tuple with a particular value for the second attribute has been assigned to $S^+$, no other tuple with the same second value can be assigned to $S^+$. This improves the decision making process by removing from consideration those tuples that violate the one-to-many constraint.

```
Repeat:
    Let $S^- = \{t_i | t_i \notin S^+ \wedge \exists t_j (t_j \in S^+ \wedge t_{i2} = t_{j2})\}$
    Let $X_1 = \{t_i | t_i \in S^? \wedge P(t_i^+ | Q_i^{+?}) > P(t_i^- | Q_i^{+?}) \wedge P(t_i^+ | Q_i^{+?}) > P(t_i^? | Q_i^{+?})\}$
    Let $X_2 = \{t_i | t_i \in X_1 \wedge \forall (t_j \in X_1) \frac{P(t_i^+ | Q_i^{+?})}{P(t_i^? | Q_i^{+?})} \geq \frac{P(t_j^+ | Q_j^{+?})}{P(t_j^? | Q_j^{+?})}\}$
    If $X_2 \neq \emptyset$ then move $X_2$ from $S^?$ to $S^+$.
Until $X_2 = \emptyset$
```

First, this algorithm constructs a new $S^-$ from $S^+$ as discussed above. Then it selects tuples from $S^?$ such that the probability that the tuple should be reassigned to $S^+$ is greater than both the probability that it should be reassigned to the new $S^-$ and the probability that it should remain in $S^?$. From this set, it selects that tuple which maximizes the ratio of the probability that it should be reassigned to $S^+$ over the probability that it should remain in $S^?$. This tuple is moved from $S^?$ to $S^+$. This process repeats until the most recent iteration has produced no changes. In effect, this algorithm iteratively moves tuples from the set of unassigned tuples into $S^+$ until no more tuples pass the criteria for being moved, each time using an $S^-$ that has been constructed using knowledge about the one-to-many nature of the target relation.

## 5 Experiment

We hypothesize that the precision and recall of an extraction algorithm will be higher if it makes use of information about the nature of the target relation. To test this hypothesis, we have chosen a relation that is one-to-many. We compare the accuracy of an algorithm developed for an arbitrary relation (many-to-many) to one developed for a one-to-many relation.

For this evaluation, we constructed the subset of documents from Wikipedia that contain the word “geography”. All sentences from those documents that contain both a country name and a continent name were parsed into dependency graphs by the Stanford NLP Group statistical parser [13]. These sentences form the corpus. The domain of possible tuples contains tuples of one country name and one continent name. The target relation relates the name of a country to the name of the continent in which it is located. It is not strictly one-to-many, since several countries lay on the border of two continents. However, the vast majority of countries are contained strictly within one continent. This relation was chosen because results in support of the hypothesis would also indicate that the domain model does not have to exactly match the real world data to have a beneficial effect on the results.

Country names were taken from the NGA GNS Names File, a freely available list of place names provided by the National Geospatial-Intelligence Agency.The continent names type consists of six continents: *America*, *Europe*, *Asia*, *Africa*, *Antarctica*, and *Oceania*. Variations in the name, such as *North America* and *Southeast Asia* are allowed and canonicalized to their shorter, more general form.

We evaluated the extracted tuples against ground truth. The ground truth used for this evaluation was a third-party description of which country exists in which continents. These values were taken from the CIA’s *The World Factbook*, a publication with political facts about countries [14]. We attempted to model the information in the CIA World Factbook as closely as possible. The Factbook does not present this relation as a one-to-many relation, though only a few countries violate this property (Russia and Azerbaijan, for instance, lie both in Asia and in Europe). In these cases, both continents were considered correct.

The algorithm was run with seed sets varying from one positive tuple to 40 positive tuples. The results from one thousand runs were averaged together.

Recall was calculated as the number of correct tuples in $S^+$ over the number of correct tuples. Precision was calculated as the number of correct tuples in $S^+$ over the total number of tuples in $S^+$. That is, we only counted tuples that were correctly identified by the algorithm as representing fact ($S^+$). We did not count tuples which the algorithm classified as counter-factual ($S^-$). We also followed [2] and did not count tuples that did not occur in the document set. We chose to calculate it this way since a naive algorithm which classified all tuples as negative would classify approximately $\frac{5}{6}$, or 83% of the tuples correctly without identifying one correct country-continent pair. The harder problem of extracting the data table containing country-continent pairs would be left unsolved. It is also unfair to penalize the algorithm for not extracting something that did not exist in the corpus.

## 6 Results

Figures 1 and 2 show the recall and precision of the two algorithms versus the seed set size (number of positive seeds given). Three notable behaviors occur in these graphs.

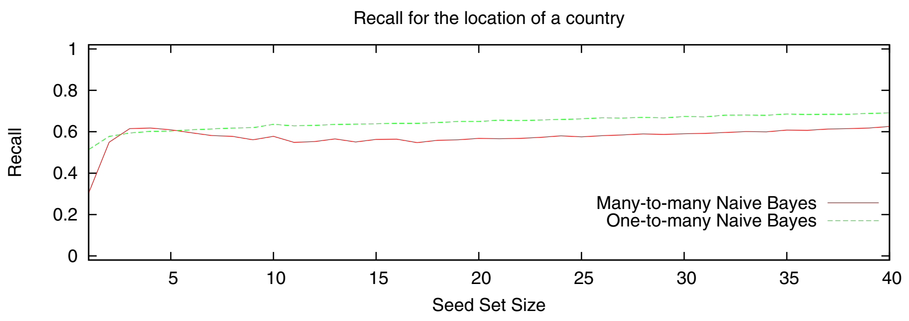


**Fig. 1.** The recall of the one-to-many algorithm slightly outperforms the unspecialized algorithm

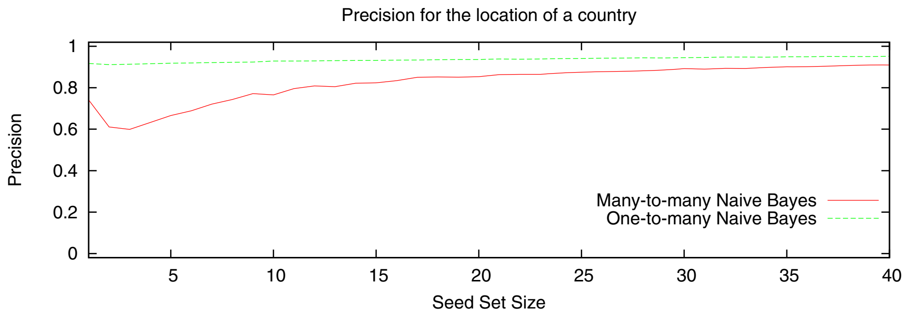


**Fig. 2.** The precision of the one-to-many algorithm takes fewer seeds to stabilize

First, in the many-to-many algorithm's performance, there is a sharp increase in the recall graph between 1 and 2 seeds. This increase comes with a concomitant decrease in precision. This phenomenon indicates that the many-to-many algorithm classified many tuples as positive, though many were not actually positive. The classification was unreliable. This behavior typifies a classifier that has insufficient knowledge about a relation and is guessing wildly. The algorithm did not begin to stabilize and achieve steady improvement until after ten seeds were inputted to it. The one-to-many algorithm did not exhibit this behavior. It improved steadily from the first through the last seed set size due to its more extensive knowledge of the nature of the relation. It should be noted that the shape of the graph could lead to false conclusions if one only examined the first several data points.

Second, the one-to-many and many-to-many graphs eventually converge. This indicates that the extra information that the one-to-many algorithm contained was eventually compensated for by the larger number of seeds. As the number of seeds given to the algorithm increased, the advantage due to exploiting properties of the relation decreased. However, for low numbers of seeds, the one-to-many algorithm outperforms the many-to-many algorithm. Better performance with fewer input seeds is clearly an improvement.

Third, the recall and precision graphs eventually reach points beyond which they only increase slowly and (almost) linearly. Beyond these points, the two algorithms did not identify any new tuples by making use of the increased number of seeds. Instead, the increase in recall and precision is due to nothing more than the addition of one more user-supplied seed in the recall and precision calculations. The greatest rate of improvement in recall, without loss of precision, is seen in the one-to-many algorithm with smaller numbers of seeds. This indicates that a small number of seeds was sufficient for the algorithm to quickly learn the majority of the relation that appears in the corpus.

When run on the same one-to-many relation, the one-to-many algorithm achieved higher and more stable accuracy with fewer seeds than the many-to-many algorithm. The results support the hypothesis that an algorithm specialized to a specific relation type performs better than an unspecialized algorithm when extracting a relation of that type.

## 7 Conclusion and Future Work

In this paper we have quantified the improvement in performance gained by a relation extraction algorithm by making use of additional information about the cardinality of a relation. The results of our experiments show that an algorithm designed to take advantage of the one-to-many property performs better, on a one-to-many relation, than an unspecialized algorithm. This work suggests that the performance of many relation extraction algorithms can be improved by incorporating knowledge about the nature of the target relation. Moreover, many properties, such as the uniqueness constraint in the one-to-many relation, are very common and highly generalizable. It is believed that an algorithm could be developed that would accept these domain properties as input, along with the seeds, to improve its extraction accuracy. Further research to discover and characterize these properties is therefore warranted.

## Acknowledgements

This work was sponsored by the U.S. Naval Research Laboratory's Base Program, Program Element No. 0602435N.

## References

1. Brin, S.: Extracting patterns and relations from the world wide web. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 172–183. Springer, Heidelberg (1998)
2. Agichtein, E., Gravano, L.: Snowball: extracting relations from large plain-text collections. In: Proceedings of the fifth ACM conference on Digital libraries, pp. 85–94 (2000)
3. Bunescu, R., Mooney, R.: A shortest path dependency kernel for relation extraction. In: Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, pp. 724–731 (2005)

4. Riloff, E.: Automatically Generating Extraction Patterns from Untagged Text. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence, vol. 2, pp. 1044–1049 (1996)
5. Agichtein, E., Eskin, E., Gravano, L.: Combining Strategies for Extracting Relations from Text Collections. In: Proceedings of the 2000 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD 2000) (May 2000)
6. Pennacchiotti, M., Pantel, P.: A Bootstrapping Algorithm for Automatically Harvesting Semantic Relations. In: Proceedings of Inference in Computational Semantics (ICoS 2006), Buxton, England (2006)
7. Pantel, P., Pennacchiotti, M.: Espresso: leveraging generic patterns for automatically harvesting semantic relations. In: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL, pp. 113–120 (2006)
8. Liu, Y., Shi, Z., Sarkar, A.: Exploiting Rich Syntactic Information for Relation Extraction from Biomedical Articles. In: Procs. of NAACL/HLT (2007)
9. Etzioni, O., Cafarella, M., Downey, D., Popescu, A., Shaked, T., Soderland, S., Weld, D., Yates, A.: Methods for Domain-Independent Information Extraction from the Web: An Experimental Comparison. In: Proceedings of the AAAI Conference, pp. 391–398 (2004)
10. Hasegawa, T., Sekine, S., Grishman, R.: Discovering Relations among Named Entities from Large Corpora. In: Proceedings of the Annual Meeting of Association of Computational Linguistics (ACL) (2004)
11. Shinyama, Y., Sekine, S.: Preemptive information extraction using unrestricted relation discovery. In: Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, pp. 304–311 (2006)
12. Culotta, A., McCallum, A.: Reducing labeling effort for structured prediction tasks. In: Veloso, M.M., Kambhampati, S. (eds.) AAAI, pp. 746–751. AAAI Press / The MIT Press (2005)
13. de Marneffe, M., MacCartney, B., Manning, C.: Generating typed dependency parses from phrase structure parses. In: LREC 2006 (2006)
14. The World Factbook. Central Intelligence Agency (2008)

# *Cor-Split*: Defending Privacy in Data Re-publication from Historical Correlations and Compromised Tuples

Daniele Riboni and Claudio Bettini

D.I.Co., Università di Milano, Italy
{riboni,bettini}@dico.unimi.it

**Abstract.** Several approaches have been proposed for privacy preserving data publication. In this paper we consider the important case in which a certain view over a dynamic dataset has to be released a number of times during its history. The insufficiency of techniques used for one-shot publication in the case of subsequent releases has been previously recognized, and some new approaches have been proposed. Our research shows that relevant privacy threats, not recognized by previous proposals, can occur in practice. In particular, we show the cascading effects that a single (or a few) compromised tuples can have in data re-publication when coupled with the ability of an adversary to recognize historical correlations among released tuples. A theoretical study of the threats leads us to a defense algorithm, implemented as a significant extension of the *m-invariance* technique. Extensive experiments using publicly available datasets show that the proposed technique preserves the utility of published data and effectively protects from the identified privacy threats.

## 1 Introduction

There are many data repositories that store time-dependent data and that require recurrent release of recently acquired data to third parties. Many papers have addressed the problem of anonymizing datasets for one-time publication ([1,2,3,4] among many others). The main defense technique consists in providing anonymity by generalizing the values of quasi-identifier (QI) attributes, so that each released tuple belongs to a group (called QI-group) having the same value for the QI attributes. This intuitively guarantees that within a group the tuple respondents cannot be distinguished. The cardinality of the group as well as the distribution of sensitive attribute values in each group are relevant parameters for the achieved anonymity. However, less attention has been given to privacy threats that can occur upon re-publication of the same database after updates have been performed. Indeed, it has been recognized that additional privacy issues arise if the adversary obtains a history of tables anonymized as described above. For example, by understanding that tuples $t_1$ and $t_2$ in different releases refer to the same (anonymous) individual, the intersection of the candidate respondents for $t_1$ and $t_2$ can lead to a privacy violation.

**Table 1.** Original and generalized microdata at the first and second release

(a) Original microdata at time $\tau_1$

| Name | Age | Gender | Zip | Disease |
|---|---|---|---|---|
| Alice | 33 | Female | 12000 | cancer |
| Betty | 31 | Female | 11000 | bronchitis |
| Carl | 35 | Male | 12000 | AIDS |
| Doris | 40 | Female | 13000 | cancer |
| Erica | 41 | Female | 14000 | AIDS |
| Fiona | 37 | Female | 13000 | bronchitis |

(b) Generalized microdata: 1st release

| QI-group | Age | Gender | Zip | Disease |
|---|---|---|---|---|
| 1 | [31,35] | × | [11k,12k] | cancer |
| 1 | [31,35] | × | [11k,12k] | bronchitis |
| 1 | [31,35] | × | [11k,12k] | AIDS |
| 2 | [37,41] | Female | [13k,14k] | cancer |
| 2 | [37,41] | Female | [13k,14k] | AIDS |
| 2 | [37,41] | Female | [13k,14k] | bronchitis |

(c) Original microdata at time $\tau_2$

| Name | Age | Gender | Zip | Disease |
|---|---|---|---|---|
| Carl | 35 | Male | 12000 | AIDS |
| Doris | 40 | Female | 13000 | cancer |
| Fiona | 37 | Female | 13000 | bronchitis |
| Erica | 41 | Female | 14000 | AIDS |
| *Grace* | 42 | Female | 13000 | bronchitis |
| *Hanna* | 42 | Female | 13000 | cancer |

(d) Generalized microdata: 2nd release

| QI-group | Age | Gender | Zip | Disease |
|---|---|---|---|---|
| 3 | [35,40] | × | [12k,13k] | AIDS |
| 3 | [35,40] | × | [12k,13k] | cancer |
| 3 | [35,40] | × | [12k,13k] | bronchitis |
| 4 | [41,42] | Female | [13k,14k] | AIDS |
| 4 | [41,42] | Female | [13k,14k] | bronchitis |
| 4 | [41,42] | Female | [13k,14k] | cancer |

As a motivating example, we consider data about patients and their cause of hospitalization (called *disease* for simplicity in the rest of this paper) frequently released by a hospital to certain institutions for data analysis. Each released table contains one tuple for each patient hospitalized during the last $L$ months. In this scenario, certain tuples may be present in multiple releases, some tuples that never appeared before can appear in new releases, and other tuples may disappear in subsequent releases. Hence, we consider updates involving both insertion and removal of tuples. For the sake of simplicity, we assume that when a tuple appears in multiple releases, the corresponding private value remains the same.[1] We consider the realistic case that some tuples may be compromised; for example, the actual disease of some patient may be known to the adversary (note that, at least, every patient is aware of her own disease). The following examples illustrate privacy threats that can occur in such a scenario.

*Example 1.* Consider the original microdata at time $\tau_1$ and $\tau_2$, shown in Tables 1(a) and 1(c), and the generalized microdata in Tables 1(b) and 1(d). Note that each generalization guarantees anonymity according to state-of-the-art techniques ($k$-anonymity [1] with $k = 3$, $l$-diversity [3] with $l = 3$, and $t$-closeness [4] with $t = 0$); moreover the generalization at time $\tau_2$ also satisfies $m$-invariance [5] with $m = 3$, a technique specifically designed for data re-publication.

The respondents of tuples belonging to QI-group 3 in Table 1(d) are Doris, Fiona and Carl, and the set of their candidate private values is {cancer, bronchitis, AIDS}. Suppose that the tuple about Carl has been compromised, hence revealing to the adversary that Carl has AIDS. This leads the adversary to derive by exclusion that either Doris was hospitalized for cancer and Fiona for bronchitis, or vice versa.

[1] The assumption, also made in [5], can be easily relaxed by associating an id to each tuple as often happens for real data. Referring to our example, we can use a different id for a new hospitalization of the same patient (possibly for a different disease) to enable the same kind of attack.

Now, note that two new tuples have been inserted at $\tau_2$, namely those regarding Grace and Hanna, while the tuples regarding Alice and Betty have been removed. In order to understand what we mean by historical correlation, consider the histories of QI-groups of Doris' and Fiona's tuples, i.e., QI-group 2 in Table 1(b) and QI-group 3 in Table 1(d). The set of respondents of the first group is {Doris, Erica,Fiona} while for the second is {Carl, Doris, Fiona}, and the corresponding set of private values is {cancer, bronchitis, AIDS} for both. By assumption, the presence in both releases of tuples for Doris and Fiona imply that each of them preserved her private value across releases. Hence, the possible private values for Doris and Fiona, considering that Carl's tuple has been compromised, were {cancer, bronchitis} even at $\tau_1$. Since at $\tau_1$ the possible private values for the QI-group 2 (including a tuple whose respondent must be Erica) were {cancer, bronchitis, AIDS}, the adversary can conclude that Erica was hospitalized for AIDS.

One of the first attempts to address privacy issues in data re-publication can be found in [6]. That work proposes a technique to preserve a weak form of $l$-diversity when multiple versions of the same table are released over time and the table is updated by insertions only. One shortcoming of that technique is that microdata publishing is postponed until the conditions for guaranteeing the required level of $l$-diversity are met. A similar scenario is addressed in [7] and in [8] in the case in which tuples are released together with their unique identifier or not, respectively. Even if the solutions proposed in those works do not require delaying data publication, they are restricted to the case in which tables are updated with insertions only, and cannot be applied when tuples are removed. The first work to address privacy-preserving data re-publication when both insertions and deletions are allowed is [5], in which the $m$-invariance property is proposed. That property ensures that *i)* all the QI-groups in which a tuple appears have the same set of private values (the cardinality of such set must be greater than or equal to $m$), and *ii)* the set of private values of QI-groups maximize the level of diversity (i.e., each QI-group does not contain tuples having the same private value). However, $m$-invariance is prone to privacy threats (as the ones exemplified above) that were not identified before, for which we propose both a theoretical study and a defense algorithm.

In this paper we consider the same scenario considered in [5], admitting both insertion and removal of tuples, but also considering the case that some tuples may be compromised. We show that, even when a very small percentage of tuples is compromised, the correlation that can be identified between tuples in different releases can lead to serious privacy leaks.

The main contributions of our work are the following: a) We perform a probabilistic analysis showing that in realistic cases the application of state-of-the-art techniques for privacy preservation in data re-publication can fail to protect the privacy of individuals. b) We propose the *Cor-Split* algorithm as a defense technique, inspired by $m$-invariance, against attacks exploiting compromised tuples and historical correlations. The algorithm is proved to correctly provide protection. c) We show experimental results on public data directly comparing

Cor-Split with $m$-invariance: From the experiments we can conclude that the extra protection offered by our algorithm has negligible costs over previously known techniques.

The rest of this paper is organized as follows. Section 2 formalizes the privacy threat we are considering. Section 3 reports a probabilistic analysis showing the actual risk of a privacy breach according to the value of some parameters. Section 4 illustrates the Cor-Split defense algorithm and its formal properties. Section 5 shows experimental results, and Section 6 concludes the paper.

## 2 Model of Privacy Threats

In this section we formally model the notions of privacy breach, and the functions that may be used by an adversary to restrict the set of private values associated to each candidate tuple respondent. We call such functions *private value restriction functions.*

### 2.1 Preliminary Definitions

In this paper we denote by $T_j$ an original table at time $\tau_j$, and by $T_j^*$ the generalization of $T_j$ released by the data publisher; we denote by $\mathcal{H}_{1,j}^* = \langle T_1^*, T_2^*, \ldots, T_j^* \rangle$ a *history* of released generalized tables. We say that a tuple $t$ has lifespan $\mathcal{L}$ if the generalization $t^*$ of $t$ appeared in each table $T_i^*$ with $i \in \mathcal{L}$; we denote by $t.r$ the respondent of $t$.

Tables are generalized by a *generalization function* $G : \mathcal{T} \times \widetilde{\mathcal{H}} \times \mathcal{R} \times \Theta \rightarrow \mathcal{T}^*$, where $\mathcal{T}$ is the set of possible original tables, $\widetilde{\mathcal{H}}$ is the set of possible histories of original microdata tables, $\mathcal{R}$ is the collection of possible sets of tuples respondents, $\Theta$ is the set of functions that map each respondent $r$ to her set of possible private values $S_r$, and $\mathcal{T}^*$ is the set of possible generalized tables. We denote by $S_{r,j}$ the set of possible private values of respondent $r$ at time $\tau_j$.

We assume that the schema of tables in $\mathcal{H}_{1,j}^*$ remains unchanged throughout the release history, and we classify the table columns into a set $A^{qi}$ of quasi-identifier attributes (for the sake of simplicity we assume that categorical values are transformed in numeric ones), and into a single private attribute $A^s$ having domain $S$; $t[A]$ is the projection of $t$ onto $A$. Columns that do not act as either quasi-identifier or private value are irrelevant with respect to privacy preservation and therefore they are ignored in the rest of the paper. Tuples in $T^*$ are partitioned into *QI-groups*; i.e., sets of tuples having the same values for their quasi-identifier attributes. We denote by $Q.R$ the set of respondents of tuples belonging to a QI-group $Q$. The *signature* $Q.sig$ of $Q$ is the set of private values of tuples belonging to $Q$.

We assume that the background knowledge available to an adversary is composed of the generalization function $G$, and the set $R$ of respondents, as well as their QI values and their sets of possible private values. Moreover, as usual in related work, we assume that, given a QI-group $Q$, an adversary may get to know the exact set of respondents of tuples in $Q$. Note that the latter is a conservative assumption, since in general the generalized QI values of tuples in $Q$ could match more users than the actual respondents of $Q$'s tuples.

**Definition 1 (privacy breach).** *A* privacy breach *occurs when an adversary knows the* sensitive association *between a user and one or more of her private values.*

## 2.2 Threats Deriving from Compromised Tuples

As shown by Example 1, the presence of a compromised tuple in a QI-group $Q$ can be used by an adversary to restrict the set of possible private values of the respondents of other tuples in $Q$. This kind of adversarial inference can be modeled as a *compromised tuples-based private value restriction (ct-pvr) function.*

**Definition 2 (ct-pvr function).** *Given a history of released tables $\mathcal{H}^*_{1,j}$, the respondent $r$ of a tuple $t^*$ in $\mathcal{H}^*_{1,j}$, $r$ having prior set of possible private values $S_r$, the set $\mathcal{H}^*_{1,j}(Q,t)$ of QI-groups published in $\mathcal{H}^*_{1,j}$ and containing a generalization of $t$, and a set $C_{\mathcal{H}^*_{1,j}}$ of compromised tuples published in $\mathcal{H}^*_{1,j}$, a* ct-pvr function *is a function ct-pvr* : $R \times 2^S \times 2^{2^{T^*}} \times 2^{T^*} \rightarrow 2^S$ *such that:*

$$\begin{aligned} &\textit{ct-pvr}(r, S_r, \mathcal{H}^*_{1,j}, C_{\mathcal{H}^*_{1,j}}) = \\ &= S_r \setminus \{a \in S \mid \exists\, Q \in \mathcal{H}^*_{1,j}(Q,t), \forall u^* \in Q, u^*[A^s] = a \Rightarrow u^* \in C\}. \end{aligned}$$

Note that in order to discard a value for a respondent of a tuple belonging to $Q$, every tuple in $Q$ having that value should be associated by the adversary to a different respondent. The example below shows how the case of a compromised tuple reported in Example 1 applies to Definition 2.

*Example 2.* Referring to Example 1, consider the history of released tables $\mathcal{H}^*_{1,2}$ corresponding to Tables 1(b) and 1(d), and the respondent $r$=Doris having set of possible private values $S_r$ = {cancer, AIDS, bronchitis}; the set $C_{\mathcal{H}^*_{1,2}}$ of compromised tuples known by the adversary includes the first tuple in Table 1(d). Hence, by applying the ct-pvr function considering the QI-group 1, the adversary can discard the value $a$=*AIDS* from $S_r$, since *AIDS* is known to be the private value of Carl's tuple. Consequently, after the application of the ct-pvr function the set of Doris' possible private values contain only *cancer* and *bronchitis*. The same reasoning can be applied with $r$=Fiona.

## 2.3 Threats Deriving from Re-published Microdata

As shown in Example 1, re-published microdata is prone to a specific class of adversarial inference. In order to model this kind of privacy threats we introduce the notion of *historical correlation.*

**Definition 3 (historical correlation).** *Given a history of released tables $\mathcal{H}^*_{1,j}$, and two QI-groups $Q_1 \subseteq T^*_i \in \mathcal{H}^*_{1,j}$ and $Q_2 \subseteq T^*_l \in \mathcal{H}^*_{1,j}$ ($i \neq l$), a historical*

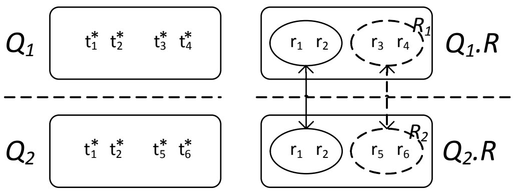


**Fig. 1.** Derivation of a historical correlation

*correlation between two sets of respondents $R_1$ and $R_2$ can be recognized if there exist two QI-groups $Q_1$ and $Q_2$ ($Q_1 \neq Q_2$ and $Q_1.S = Q_2.S$, where $Q_1.S$ and $Q_2.S$ are the multisets composed of private values of tuples in $Q_1$ and in $Q_2$, respectively) such that all the following conditions hold:*

c1) $Q_1.R \supset R_1$
c2) $Q_2.R \supset R_2$
c3) $Q_1.R \setminus R_1 \ = \ Q_2.R \setminus R_2$

*where $Q_1.R$ and $Q_2.R$ are the sets of respondents of tuples belonging to $Q_1$ and $Q_2$, respectively.*

**Theorem 1 (historical correlations).** *If two sets of respondents $R_1$ and $R_2$ are in a historical correlation, then the multiset composed of the private values of respondents in $R_1$ is equal to the multiset composed of the private values of respondents in $R_2$.*

As it was shown in Example 1, historical correlations can be used to narrow the set of candidate private values of a respondent, possibly leading to the exact derivation of her private value. Such correlations are called *historical* since they rely on the presence of the same sets of tuples in multiple views belonging to a history of releases. For instance, in Example 1 an adversary was able to find a historical correlation between $R_1$={Carl} and $R_2$={Erica} by observing QI-group 2 and QI-group 3, released at time $\tau_1$ and $\tau_2$, respectively:

c1) (QI-group 3).$R \supset$ {Carl}
c2) (QI-group 2).$R \supset$ {Erica}
c3) (QI-group 3).$R$ $\setminus$ {Carl} = (QI-group 2).$R$ $\setminus$ {Erica}

Similarly, a historical correlation between {Alice, Betty} and {Doris, Fiona} could be discovered observing QI-group 1 and QI-group 3.

*Example 3.* Consider Figure 1, which depicts two QI-groups $Q_1 = \{t_1^*, t_2^*, t_3^*, t_4^*\}$ and $Q_2 = \{t_1^*, t_2^*, t_5^*, t_6^*\}$, and the sets of respondent of $Q_1$ and $Q_2$, which are $Q_1.R = \{r_1, r_2, r_3, r_4\}$ and $Q_2.R = \{r_1, r_2, r_5, r_6\}$, respectively. In this situation a historical correlation between $R_1 = \{r_3, r_4\}$ and $R_2 = \{r_5, r_6\}$ can be recognized. Indeed, the set of respondents $\{r_1, r_2\}$ (enclosed in a solid ellipse) appears in both $Q_1.R$ and $Q_2.R$ and, since the private value of each tuple cannot change, the private values of $r_1$ and $r_2$ are the same in $Q_1$ and in $Q_2$. As a consequence, the set of private values of $\{r_3, r_4\}$ is the same as that of $\{r_5, r_6\}$ (the sets of tuples related by a historical correlation are enclosed in a dashed ellipse).

The adversarial inference that exploits historical correlations to the aim of restricting the set of possible private values of a tuple respondent can be modeled according to the following *historical correlation-based private value restriction (hc-pvr) function.*

**Definition 4 (hc-pvr function).** *Given a history of released tables $\mathcal{H}^*_{1,j}$, the respondent $r$ of a tuple $t^*$ in $\mathcal{H}^*_{1,j}$, $r$ having initial set of possible private values $S_r$, and the set $\mathcal{R}(\mathcal{H}^*_{1,j}, r)$ of sets of respondents that are linked to $r$ by a historical correlation in $\mathcal{H}^*_{1,j}$, a* hc-pvr function *is a function hc-pvr* $: R \times 2^S \times 2^{2^R} \to 2^S$ *such that:*

$$\textit{hc-pvr}(r, S_r, \mathcal{R}(\mathcal{H}^*_{1,j}, r)) = S_r \setminus \{a \in S \mid \exists\, R \in \mathcal{R}(\mathcal{H}^*_{1,j}, r), \forall r' \in R,\ a \notin S_{r',j}\}.$$

The following example shows how the adversarial inference presented in Example 1 applies to Definition 4.

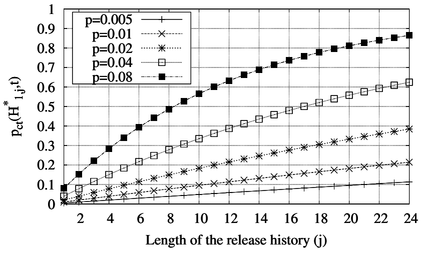
*Example 4.* Referring to Example 1, consider the history of released tables $\mathcal{H}^*_{1,2}$ corresponding to Tables 1(b) and 1(d), and the respondent $r$=Erica having set of possible private values $S_r$ = {cancer, AIDS, bronchitis}; $\mathcal{R}(\mathcal{H}^*_{1,2}, Erica)$ includes the set {Carl} (i.e., a historical correlation relating Erica and Carl was discovered). The set of Carl's possible private values includes neither *cancer* nor *bronchitis* (i.e., the adversary knows that the private value of his tuple is *AIDS*). Hence,

$$\begin{aligned}&\text{hc-pvr}(Erica, S_{Erica}, \mathcal{R}(\mathcal{H}^*_{1,2}, Erica)) =\\ &\qquad = \{\text{cancer}, \text{AIDS}, \text{bronchitis}\} \setminus \{\text{cancer}, \text{bronchitis}\} = \{\text{AIDS}\}.\end{aligned}$$

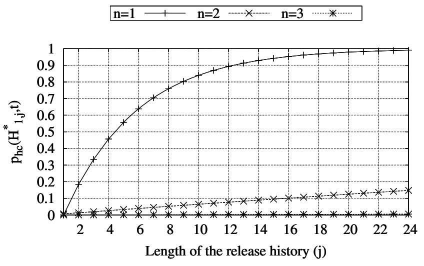
Then, after the second release the adversary derives that the set of possible private values of Erica is {AIDS}. As a consequence, the sensitive association between Erica and AIDS is discovered.

## 3 Probabilistic Analysis

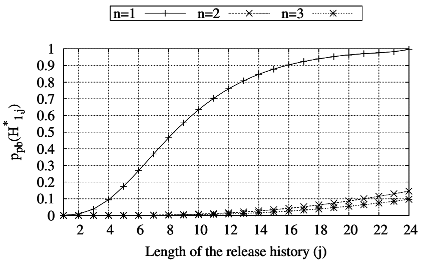
The actual risk of a privacy breach due to the threats we are considering depends on several parameters. In this section we perform a probabilistic analysis of this risk, assuming that each tuple has probability $p$ of being compromised. Without loss of generality, we also assume that released microdata satisfy the $m$-invariance principle, since – as it will be shown later in this section – this is a worst case for our probabilistic analysis. As a consequence, we also assume that the set $S$ of private values includes at least as many values as the enforced level $m$ of $m$-invariance. We also assume that each tuple is released at most $L$ times; i.e., for each tuple $t$, $L$ is an upper limit for the cardinality of its lifespan $\mathcal{L}$. The probability of privacy breach is measured, depending on the parameters $p$, $m$, $L$ as well as others, as the result of the application of the private value restriction functions described in Sections 2.2 and 2.3.



(a) $p_{ct}(\mathcal{H}^*_{1,j}, t)$



(b) $p_{hc}(\mathcal{H}^*_{1,j}, t)$ with $p = 0.04$ and $m = 6$



(c) $p_{pb}(\mathcal{H}^*_{1,j}, t)$ with $p = 0.04$ and $m = 6$

**Fig. 2.** Probabilistic analysis

## 3.1 Probability of Excluding Private Values Due to Compromised Tuples

As illustrated in Section 2.2, given a QI-group $Q$, if all the tuples in $Q$ having private value $a$ are compromised, then an adversary can conclude that no other respondent of tuples in $Q$ has private value $a$. Of course, the probability that such an event occurs decreases the higher is the number of occurrences of tuples in $Q$ having private value $a$. Hence, in order to analyze the worst case we assume that each private value is owned by at most one tuple in a single QI-group. This property is called $m$-uniqueness in [5], and it is guaranteed by the enforcement of the $m$-invariance principle. The corresponding privacy threat is quantified by the following lemma.

**Lemma 1.** *Given a history of released tables $\mathcal{H}^*_{1,j}$ satisfying the m-invariance principle, a set $C_{\mathcal{H}^*_{1,j}}$ of compromised tuples belonging to $\mathcal{H}^*_{1,j}$, and a tuple t having respondent r and lifespan $\mathcal{L}_{\mathcal{H}^*_{1,j}}$ in $\mathcal{H}^*_{1,j}$ with $\max\{|\mathcal{L}_{\mathcal{H}^*_{1,j}}|\} \leq j$, the probability that a private value is discarded from the set $S_r$ of possible private values of r through the application of function ct-pvr$(r, S_r, \mathcal{H}^*_{1,j}, C_{\mathcal{H}^*_{1,j}})$ is:*

$$p_{ct}(\mathcal{H}^*_{1,j}, t) = 1 - (1-p)^{|\mathcal{L}_{\mathcal{H}^*_{1,j}}|},$$

*where p is the probability of a generic tuple to be compromised.*

The plot shown in Figure 2(a) shows the value of $p_{ct}(\mathcal{H}^*_{1,j}, t)$ with respect to the number of releases containing the tuple (here we assume that the lifespan of $t$

covers the entire history) for different values of $p$. As expected, $p_{ct}(\mathcal{H}^*_{1,j}, t)$ grows with the length of the release history and with the value of $p$.

## 3.2 Probability of Excluding Private Values Due to Historical Correlation

The probability that a private value is excluded from the set of candidate private values due to historical correlations is given by the following lemma.

**Lemma 2.** *Given a history of released tables $\mathcal{H}^*_{1,j}$ satisfying the $m$-invariance property, a tuple $t$ having respondent $r$ and lifespan $\mathcal{L}_{\mathcal{H}^*_{1,j}}$ in $\mathcal{H}^*_{1,j}$ with $\max\{|\mathcal{L}_{\mathcal{H}^*_{1,j}}|\} \leq j$, and the set $\mathcal{R}(\mathcal{H}^*_{1,j}, r)$ of sets of respondents that are linked to $r$ by a historical correlation in $\mathcal{H}^*_{1,j}$, the probability that a private value $a$ is discarded from the set $S_r$ of possible private values of $r$ through the application of function hc-pvr$(r, S_r, \mathcal{R}(\mathcal{H}^*_{1,j}, r))$ is:*

$$p_{hc}(\mathcal{H}^*_{1,j}, t) = 1 - \left(1 - \left(p - \frac{p}{m}\right)^n\right)^{|\mathcal{L}_{\mathcal{H}^*_{1,j}}| \cdot \lfloor \frac{m}{n} \rfloor},$$

*where $n = \min_{R \in \mathcal{R}(\mathcal{H}^*_{1,j}, r)} \{|R|\}$ is the minimum cardinality of sets of respondents in $\mathcal{R}(\mathcal{H}^*_{1,j}, r)$, and $p$ is the probability of a generic tuple to be compromised.*

The plot shown in Figure 2(b) shows the value of $p_{hc}(\mathcal{H}^*_{1,j}, t)$ with respect to the length $j$ of the release history (assuming the lifespan of $t$ covers the entire history) and to the minimum cardinality $n$ of sets of respondents involved in historical correlations. The released tables are assumed to satisfy the $m$-invariance property with $m = 6$, and the probability of a tuple to be compromised is set to $p = 0.04$. As expected, $p_{hc}(\mathcal{H}^*_{1,j}, t)$ grows with the length of the release history of the tuple and it is higher for smaller values of $n$.

## 3.3 Probability of Privacy Breach Due to Combined Threats

After having separately considered the threats deriving from functions ct-pvr and hc-pvr we can quantify the probability of privacy breach deriving from the application of both functions.

**Theorem 2 (probability of privacy breach).** *Given a history of released tables $\mathcal{H}^*_{1,j}$ satisfying the $m$-invariance principle, a tuple $t$ having respondent $r$ and lifespan $\mathcal{L}_{\mathcal{H}^*_{1,j}}$ in $\mathcal{H}^*_{1,j}$ with $\max\{|\mathcal{L}_{\mathcal{H}^*_{1,j}}|\} \leq j$, and the set $\mathcal{R}(\mathcal{H}^*_{1,j}, r)$ of sets of respondents that are linked to $r$ by a historical correlation in $\mathcal{H}^*_{1,j}$, if $n = \min_{R \in \mathcal{R}(\mathcal{H}^*_{1,j}, r)} \{|R|\}$ is the minimum cardinality of sets of respondents in $\mathcal{R}(\mathcal{H}^*_{1,j}, r)$, and $p$ is the probability of a generic tuple to be compromised, then the probability that a privacy breach is determined by functions ct-pvr and hc-pvr at time $j$ is:*

$$p_{pb}(\mathcal{H}^*_{1,j}, t) = \left(1 - (1-p)^{|\mathcal{L}_{\mathcal{H}^*_{1,j}}|} \cdot \left(1 - \left(p - \frac{p}{m}\right)^n\right)^{|\mathcal{L}_{\mathcal{H}^*_{1,j}}| \cdot \lfloor \frac{m}{n} \rfloor}\right)^{m-1}.$$

The plot shown in Figure 2(c) shows the value of $p_{pb}(\mathcal{H}^*_{1,j}, t)$ with respect to the length $j$ of the release history (assuming the lifespan of $t$ covers the entire history) and to the minimum cardinality $n$ of sets of respondents involved in historical correlations. The released tables are assumed to satisfy the $m$-invariance property with $m = 6$, and the probability of a tuple to be compromised is set to $p = 0.04$. It can be observed that the probability of privacy breach is very high with $n = 1$. The value of $p_{pb}$ is lower than 0.15 with $n = 2$; with $n = 3$ the probability of privacy breach is lower than 0.1.

## 4 Defense

### 4.1 Safety against Private Value Restriction Functions

In order to defend re-published microdata against private value restriction functions based on compromised tuples and historical correlations, our technique consists in enforcing a generalization principle – which we name *$(m, n)$-historical safety* – with parameters that guarantee that the probability of privacy breach is below a certain threshold $h$. Before defining $(m, n)$-historical safety it is necessary to introduce a novel principle, which we name *weak $m$-invariance*. As it will we shown in Section 4.2, this principle can be applied to avoid the disclosure of historical correlations while minimizing the number of counterfeits.

**Definition 5 (weak $m$-uniqueness).** *A generalized table $T^*_j$ satisfies* weak $m$-uniqueness *if each QI-group $Q \subseteq T^*_j$ contains at least $m$ tuples with different private values, and the number of occurrences in $Q$ of tuples with a given private value is the same for every private value belonging to the signature of $Q$.*

**Definition 6 (weak $m$-invariance).** *A history of released tables $\mathcal{H}^*_{1,j}$ satisfies* weak $m$-invariance *if:*

- $\forall i \in [1, j]$, *$T^*_i$ satisfies weak $m$-uniqueness, and*
- $\forall i, i' \in [1, j]$, *$t \in T_i, t' \in T_{i'}$, if $t^* \in Q_i$ and $t'^* \in Q_{i'}$, then $Q_i.sig = Q_{i'}.sig$.*

Weak $m$-invariance is a weaker version of the $m$-invariance principle. Indeed, while $m$-invariance requires that all the tuples in a QI-group have different private values, according to weak $m$-invariance QI-groups can contain tuples with duplicate private values, provided that their multiplicity is the same. Hence, it is easy to verify that $m$-invariance is a particular case of weak $m$-invariance in which the multiplicity of tuples having a given private value in a given QI-group is always 1. With respect to privacy preservation, it can be observed that weak $m$-invariance provides the same level of diversity as the one provided by $m$-invariance. On the other hand, the obvious shortcoming of having multiplicities of private values greater than 1 in a QI-group is that in most cases the degree of generalization of QI values of that QI-group would grow with the multiplicity. Hence, as it will be shown in Section 4.2, an objective of our devised generalization algorithm is to minimize the multiplicity of private values in QI-groups.

**Definition 7 (hc-safety).** *Given a history of released tables $\mathcal{H}^*_{1,j}$ and a QI-group $Q \subseteq T^*_i \in \mathcal{H}^*_{1,j}$, $Q$ is* hc-safe with degree *$n$ if either:* i) *no set of respondents is related with the respondents of tuples in $Q$ by a historical correlation in $\mathcal{H}^*_{1,j}$, or* ii) *the cardinality of each set of respondents that is related with the respondents of tuples in $Q$ by a historical correlation in $\mathcal{H}^*_{1,j}$ is greater than or equal to $n$.*

**Definition 8 ($(m, n)$-historical safety).** *Given $m, n \in \mathbb{N}$, $n \leq m$, a generalization function $G$ is* $(m, n)$-historically safe *if, for each table $T_{j+1}$, for each history of released tables $\mathcal{H}^*_{1,j}$ satisfying weak $m$-invariance, and*

$$T^*_{j+1} = G(T_{j+1}, \mathcal{H}^*_{1,j}, R, \vartheta)$$

*($R$ is the of tuples respondents, and $\vartheta$ is the function that maps each respondent in $R$ into her set of possible private values), the following conditions hold:*

i) *$\langle \mathcal{H}^*_{1,j}, T^*_{j+1} \rangle$ satisfies weak $m$-invariance;*
ii) *each QI-group $Q \subseteq T^*_{j+1}$ is hc-safe with degree $n$ with respect to $\langle \mathcal{H}^*_{1,j}, T^*_{j+1} \rangle$.*

The above definition states that, in order to be $(m, n)$-historically safe, a generalization function must *i)* preserve weak $m$-invariance and *ii)* generate QI-groups such that the cardinality of sets involved in historical correlations that can be derived from them is greater than or equal to $n$. Condition *i)* is imposed to protect against the attacks identified in [5]; condition *ii)* is imposed to protect against historical correlations.

In order to guarantee that the probability of privacy breach for a tuple $t$ is below a certain threshold $h$ it is necessary to have an estimate of the probability $p$ of released tuples to be compromised, and to determine the maximum cardinality $L$ of its lifespan (i.e., the maximum number of times that $t$ can be republished). Note that $L$ is an upper bound for the cardinality of $\mathcal{L}_{\mathcal{H}^*_{1,j}}$ shown in the definitions of private value restriction functions (see Section 3). In general, the values of $p$ and $L$ depend on the domain of the data. For instance, a hospital releasing microdata about patients and diseases may estimate that an adversary may get to know the sensitive association about no more than 4% of its patients (hence, $p = 0.04$), and it can decide to republish each tuple at most 24 times (hence, $L = 24$). Once values for $p$ and $L$ have been determined it is possible to express the concept of safety of a generalization function against a threshold $h$.

**Definition 9 (pvr-safe generalization function).** *A generalization function $G$ is* pvr-safe *with threshold $h \in (0, 1]$ if, for any history $\mathcal{H}^*_{1,j}$ of tables generalized by $G$, $p_{pb}(\langle T^*_1, \ldots, T^*_i \rangle, t) < h$ for each $i \in [1, j]$ and for each tuple $t \in T_i$.*

## 4.2 The *Cor-Split* Algorithm

Given parameters $p$ and $L$, the chosen level $m$ of weak $m$-invariance to be enforced, and the required threshold $h$, the goal of the algorithm proposed in this paper is to enforce $(m, n)$-historical safety with the smallest possible value of

```
Input: Parameters $p$, $L$, $m$, $h$.
Output: Parameter $n$.
 1: $f(p,L,m,n) = \left(1-(1-p)^L \cdot \left(1-\left(p-\frac{p}{m}\right)^n\right)^{L\cdot\lfloor\frac{m}{n}\rfloor}\right)^{m-1}$
 2: if $\nexists\, n' \in [1,m] \mid f(p,L,m,n') < h$ then
 3:     $\overline{n} := -1$
 4: else
 5:     $\overline{n} := \min(n' \in \mathbb{N}^+ \mid f(p,L,m,n') < h)$
 6: end if
 7: return $\overline{n}$
```

**Fig. 3.** The *n-Choose* algorithm for determining the value of $n$

```
Input: $T_{j+1}$ is the microdata table at time $\tau_{j+1}$; $\mathcal{H}^*_{1,j}$ is the history of released tables; $\mathcal{H}$ is the history of original tables; $R$ is the set of tuples respondents; $\vartheta$ is the function that maps each respondent in $R$ into her set of possible private values; $m,n \in \mathbb{N}$ are the parameters for historical safety; $A^{qi}$ is the set of QI attributes.
Output: the generalized table $T^*_{j+1}$.
 1: $T^*_{j+1} := \emptyset$
 2: $S_{\text{-}} := \{t \in T_{j+1} \mid \forall\, T \in \mathcal{H},\, t \notin T\}$
 3: $S_{\cap} := T_{j+1} \setminus S_{\text{-}}$
 4: $\mathcal{B} := \text{Division}(S_{\cap})$
 5: for all buckets $B \in \mathcal{B}$ do
 6:     Balancing$(B, S_{\text{-}})$
 7: end for
 8: $\mathcal{B}' := \text{Assignment}(\mathcal{B}, S_{\text{-}}, m, \vartheta)$
 9: for all buckets $B \in \mathcal{B}'$ do
10:     $T^*_{j+1} := \text{Cor-Partition}(T^*_{j+1}, B, A^{qi}, n)$
11: end for
12: return $T^*_{j+1}$
```

**Fig. 4.** The *Cor-Split* algorithm

$n$ that guarantees that the probability of privacy breach is below $h$. Since we assume that those parameters do not change during the release history, the parameter $n$ is chosen before the generalization of the first table, and it remains unchanged throughout the release history. The algorithm for choosing $n$ is shown in Figure 3. Note that for certain values of $p$, $L$, $m$, and for a required threshold $h$, the probability of privacy breach could be higher than $h$ for every possible value of $n$. In this case, microdata would not be released unless the value of parameters $L$ or $m$ are changed. In the other case, microdata are generalized using the value of $n$ determined by the algorithm in Figure 3. Our devised generalization algorithm, shown in Figure 4, is a significant modification of the algorithm for $m$-invariant generalization proposed in [5]. Note that the algorithm in [5], though enforcing weak $m$-invariance, does not provide guarantees about the cardinality of sets of respondents involved in historical correlations. Moreover, our empirical study (reported in Section 5) shows that QI-groups generated by that algorithm may allow an adversary to derive several historical correlations between small sets of respondents, determining severe privacy threats.

**Overview of our generalization algorithm.** Given the original table $T_{j+1}$ at time $\tau_{j+1}$, the history $\mathcal{H}^*_{1,j}$ of generalized tables published before $\tau_{j+1}$, the set $R$ of respondents, and function $\vartheta$, the output of our generalization algorithm is the generalized table $T^*_{j+1}$. Our algorithm can be roughly divided into 4 phases. While the first 3 phases are essentially identical to the ones of the algorithm in [5], Phase 4 is different, since in that phase $(m, n)$-historical safety is enforced. Adopting the notation of [5] we call $S_\cap$ the set of tuples in $T_{j+1}$ that have been released before $\tau_{j+1}$, and $S_-$ the remaining tuples in $T_j$.

- **Phase 1: Division.** This phase consists in partitioning the set of tuples in $S_\cap$ into *buckets*. Each bucket is uniquely identified by a signature among the ones of tuples in $S_\cap$, and it contains only tuples that appeared in $\mathcal{H}^*_{1,j}$ in QI-groups having the same signature of the bucket.
- **Phase 2: Balancing.** The balancing phase is applied in turn to each bucket. Its goal is to guarantee that every private value of the bucket's signature is represented by the same number of tuples in the bucket. Buckets are balanced by inserting tuples belonging to $S_-$ as long as this is possible; if no other tuples in $S_-$ can be used to balance the bucket, counterfeit tuples are inserted.
- **Phase 3: Assignment.** In this phase, the remaining tuples in $S_-$ are assigned to the existing buckets as long as they remain balanced. If no other tuple can be assigned to the existing buckets without violating balancing, new buckets are created, and the remaining tuples are assigned to the new buckets such that the new buckets are balanced. The cardinality of the signature of new buckets is greater than or equal to $m$.
- **Phase 4: Cor-Partition** This phase is applied in turn to each bucket. In this phase, buckets are partitioned into weak $m$-invariant QI-groups such that, if a novel historical correlation can be identified by matching the new QI-groups with those released during $\mathcal{H}^*_{1,j}$, then the cardinality of the sets of respondents involved in it is greater than or equal to $n$; i.e., QI-groups are hc-safe with degree $n$. For brevity, when the degree $n$ of hc-safety is clear, we say that a QI-group is hc-safe (or hc-unsafe), omitting the degree of hc-safety. This phase is described in detail in the following of this section.

**Cor-Partition.** The algorithm pseudo-code is illustrated in Figure 5. Consider a generic bucket $B$ composed of $s \cdot l$ tuples ($s \geq m$), where $s$ is the cardinality of the signature of $B$ (named $B.sig$), and $l \geq 1$. After an initialization phase (line 1), the algorithm creates, for each QI attribute in $A^{qi}$, a list of the tuples in $B$ partially ordered according to their value for that attribute (line 2). We denote $L_i$ the list regarding attribute $A^{qi}_i$, and $\overline{L}$ the set of such lists. Then, a cycle is repeated until every tuple in $B$ is assigned to a QI-group (lines 3 to 20).

At first, each list is traversed in turn to obtain a weak $m$-invariant QI-group $Q_i$ (lines 6 to 8) by selecting, for each private value belonging to the signature of the

```
Input: parameters $T^*_{j+1}, B, A^{qi}, n$ obtained from the Cor-Split algorithm.
Output: $T^*_{j+1}$ incremented with the anonymization of tuples in $B$.
 1: int $c := 0$; $\overline{L} := \emptyset$; $Q_i^{(old)} := \emptyset$
 2: for all $A_i^{qi} \in A^{qi}$ do: List $L_i :=$ order($B$, $A_i^{qi}$); $D_i := \emptyset$; $\overline{L} := \overline{L} \cup \{L_i\}$
 3: repeat
 4:    $\overline{Q} := \emptyset$; $\overline{SP} := \emptyset$
 5:    for all $L_i \in \overline{L}$ do
 6:       repeat
 7:          QI-group $Q_i :=$ createQIG$(L_i, D_i, B.sig)$
 8:       until hcSafe$(Q_i, n, j) \vee (|Q_i| < |B.sig|)$
 9:       if $|Q_i| < |B.sig|$ then: $Q_i :=$ createQIG$(L_i, \emptyset, B.sig)$; $Q_i^{(temp)} := Q_i$; $c' := c$
10:       while $\big(\neg$ hcSafe$(Q_i, n, j)\big) \wedge \big(c' > 0\big)$ do
11:          $c'_i := c' - 1$; $Q_i := Q_i \cup QIG_{c'}$
12:       end while
13:       if $\neg$ hcSafe$(Q_i, n, j)$ then: $Q_i := Q_i^{(temp)}$; $Q_i$.setCounterfeits()
14:    end for
15:    $i' := i \in \mathbb{N} \,|\, Q_i.sp = \min_{\forall j \in \mathbb{N}} \{Q_j.sp\}$
16:    $QIG_c := Q_{i'}$; $T^*_{j+1}$.removeDuplicates$(QIG_c)$ $T^*_{j+1} := T^*_{j+1} \cup QIG_c$
17:    $B := B \setminus QIG_c$; $c := c + 1$
18:    for all $L_i \in \overline{L}$ do: $L_i := L_i$.remove$(QIG_c)$; $D_i := \emptyset$
19: until $B = \emptyset$
20: return $T^*_{j+1}$
```

**Fig. 5.** The *Cor-Partition* algorithm

bucket, the first tuple in $L_i$ having that value, temporarily discarding those tuples $L_i$ that would determine hc-unsafe QI-groups. Temporarily discarded tuples are randomly chosen from tuples in $Q_i$ and replaced with other tuples in $B$ having the same private value, as long as either the resulting QI-group is hc-safe, or no more tuples are available from $B$. In the latter case (lines 9 to 12), an hc-unsafe QI-group $Q_i^{(temp)}$ is created, and it is merged with a growing number of QI-groups previously created from the same bucket, until either the resulting QI-group is hc-safe, or no other available QI-group remains. In the latter case (line 13), $Q_i^{(temp)}$ is transformed by substituting a growing number of tuples in it with counterfeits, until the resulting QI-group $Q_i$ is hc-safe. Note that counterfeit tuples are inserted only in the case in which no other operation is possible to generate a hc-safe QI-group.

Hence, after line 14, for each QI attribute in $A^{qi}$ a hc-safe QI-group is available. For each of these QI-groups we call *semiperimeter* the sum of the normalized lengths of the interval of each QI value of tuples in it. Obviously, smaller semiperimeters correspond to finer-grained generalization. For this reason, the QI-group $Q_j$ having the smallest semiperimeter is chosen (line 15). Then (lines 16 to 18), tuples in $Q_j$ are removed from $B$, as well as from the lists in $\overline{L}$, and are added to $T^*_{j+1}$, after having possibly removed duplicate tuples (indeed, in line 11, $Q_j$ could have been merged with QI-groups already inserted into $T^*_{j+1}$). After that, if the bucket contains other tuples the algorithm continues from line 4; otherwise it returns the generalized table $T^*_{j+1}$.

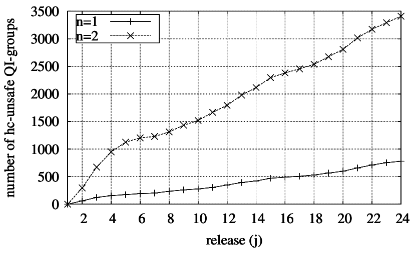
```
Input: parameters $Q, n, j$ obtained from the Cor-Split algorithm.
Output: true if $Q$ is hc-safe with degree $n$; false otherwise.
 1: for int $i := 0$ to $j$ do
 2:    multiset $(C, \omega) := multiset(\emptyset, \omega)$
 3:    for all generalized tuple $t^* \in Q$ do
 4:       respondent $r := t^*.r$
 5:       original tuple $t := r.t$
 6:       QI-group $Q' := t.QI(i)$
 7:       if $Q' \neq null$ then
 8:          $C := C \cup Q'.id$
 9:       end if
10:    end for
11:    $l := \max_{c \in C}(\omega(c))$
12:    if $|Q| - n < l < |Q|$ then
13:       return false
14:    end if
15: end for
16: return true
```

**Fig. 6.** Algorithm *hcSafe* for checking the hc-safety of a QI-group

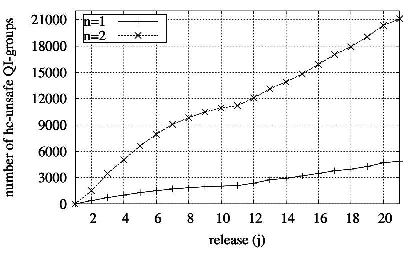
**Checking hc-safety.** The algorithm for checking the hc-safety of a QI-group to be released in $T^*_{j+1}$ is illustrated in Figure 6. Given a release history $\mathcal{H}^*_{1,j}$, a QI-group $Q$ and a degree $n \in \mathbb{N}$, it follows from Definition 3 that in order to check whether $Q$ is hc-unsafe it is sufficient to check whether it exists a set of $l$ respondents whose tuples belonged to the same QI-group both in release $T^*_j$ and $T^*_i \in \mathcal{H}^*_{1,j}$, with $l$ smaller than $|Q|$ and greater than $|Q| - n$. Hence (lines 2 to 10), for each release $T^*_i \in \mathcal{H}^*_{1,j}$ the algorithm creates a multiset that contains, for each respondent $r$ of tuples in $Q$, the unique identifier of the QI-group that included $r$'s tuple in release $T^*_i$ (if it exists). Then (line 11), the maximum multiplicity $l$ of the elements of the multiset is calculated; i.e., $l$ is the maximum number of respondents of tuples in $Q$ whose tuples also belonged to the same QI-group in $T^*_i$. If it exists at least one release $T^*_i$ such that the value $l$ is smaller than $|Q|$ and greater than $|Q| - n$ the algorithm determines that $Q$ is hc-unsafe with respect to the degree $n$; otherwise $Q$ is hc-safe with degree $n$.

**pvr-safety.** The following lemma states a sufficient condition to ensure that a generalization function is pvr-safe.

**Lemma 3 (sufficient condition for pvr-safety).** *Let $p$ be the probability of released tuples to be compromised, $L$ the maximum number of times that a single tuple can be republished, $h \in (0, 1]$ the threshold for pvr-safety, and $m \in \mathbb{N}^+$ the required level of weak $m$-uniqueness. Then, if $G$ is a generalization function enforcing $(m, n)$-historical safety with $n =$* n-Choose$(p, L, m, h)$*, then $G$ is* pvr-safe *with threshold $h$.*



(a) $m = 6$, $|T^*| = 60,000$



(b) $m = 6$, $|T^*| = 200,000$

**Fig. 7.** Number of hc-unsafe QI-groups generated by the $m$-invariance algorithm

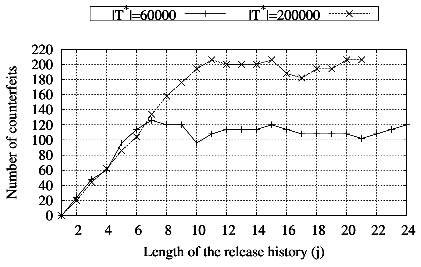
The soundness of the *Cor-Split* algorithm is proved by the following theorem.

**Theorem 3 (pvr-safety of the *Cor-Split* algorithm).** *The* Cor-Split *algorithm computes a pvr-safe generalization function.*

## 5 Experimental Evaluation

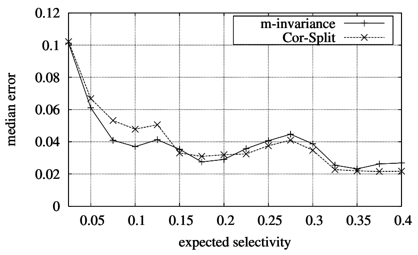
Experiments were performed using a real census dataset published by the *Minnesota Population Center* and available at `http://ipums.org/`. The dataset is composed of 600,000 tuples. Each tuple stores information about an individual; it includes 4 QI-attributes (age, birthplace, education, gender) and one private attribute *income* having 50 possible values, each one representing an income range. In order to evaluate our technique with respect to different scenarios, we simulated insertions and deletions from the dataset at different rates. Hence, we started with a table $T_1$ including 200,000 (resp. 60,000) tuples, and we obtained a table $T_2$ by randomly deleting 10% (resp. 33%) of $T_1$'s tuples and inserting the same number of tuples randomly chosen from unpublished tuples. The same procedure was repeated with the subsequent tables to obtain a history having length 21 (resp. 24). In these experiments we assumed that the probability of a tuple to be compromised is $p = 0.04$, the enforced level of (weak) $m$-invariance is $m = 6$, the maximum length of the release history of each tuple is $L = 21$ (resp. $L = 24$), and the safety threshold is $h = 0.1$. Given these parameters, the level $n$ of $(m, n)$-historical safety to be enforced is $n = 3$. Results of experiments with different values for parameters $m$ and $n$ ($m = 4 \div 10$, $n = 1 \div 5$) are not reported here for lack of space; however, they essentially lead to the same conclusions as the ones reported below.

**Historical correlations determined by *m*-invariance.** The first set of experiments aimed at evaluating the threat determined by private value restriction functions when microdata are generalized applying the $m$-invariance technique. In particular, we adopted the algorithm in [5] to generalize microdata tables, and at any release we counted the number of released QI-groups that were hc-unsafe with respect to the degree $n = 3$; hc-unsafe QI-groups were recognized using the algorithm reported in Figure 6. Results in the considered scenarios are illustrated

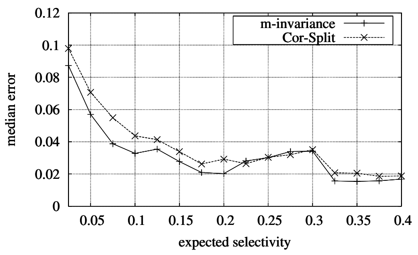


(a) $m = 6$, $n = 3$

**Fig. 8.** Number of counterfeits introduced by the *Cor-Split* algorithm



(a) $m = 6$, $n = 3$, $|T^*| = 60000$



(b) $m = 6$, $n = 3$, $|T^*| = 200000$

**Fig. 9.** Query error

in Figure 7, and show that many released QI-groups may allow an adversary to derive historical correlations between small sets of respondents (having cardinality 1 or 2), determining relevant privacy threats.

**Counterfeits and query error.** The second set of experiments was performed on microdata generalized by a Java implementation of the *Cor-Split* algorithm. Tuples were generalized in order to enforce $(m, n)$-historical safety with $m = 6$ and $n = 3$. At first, we measured the number of counterfeit tuples introduced by *Cor-Split*. Results are illustrated in Figure 8 and show that in both scenarios the algorithm introduced a few counterfeits. Then, we compared the utility of microdata generalized by *Cor-Split* and by the algorithm for $m$-invariance in terms of the precision in answering aggregate queries (e.g., *count the number of individuals in the table whose QI-values belong to certain ranges*). Queries were randomly generated according to different values of *expected selectivity*, i.e., expected ratio of tuples to be returned by the query. For each value of expected selectivity, 10,000 queries were randomly generated. The imprecision in query answering was calculated in terms of the median error of query answers. The results reported in Figure 9 show that the accuracy of query answering obtained by *Cor-Split* is very close the to one observed with the use of the generalization algorithm for $m$-invariance, with the advantage of protecting from the threats we have identified.

## 6 Conclusions and Future Work

In this paper we have addressed privacy threats that may arise when a certain view over a dynamic dataset has to be released multiple times during its history. We have shown the limits of existing techniques to protect privacy in the case an adversary is able to recognize correlations between sets of tuples released in different views and even a small percentage of tuples is compromised. After having formalized the problem, we have provided a probabilistic study of the identified threats, and we have proposed a sound defense algorithm that has been experimentally validated. Future work includes the study of other attacks based on correlation between different releases. In particular, we are currently investigating the case in which tuples for the same respondent in different releases can have different private values; in this scenario our defense can still be effective against the attacks considered in this paper, but the adversary may exploit a different kind of historical correlation, based on private values associated to candidate respondents in a history of released tuples.

## Acknowledgments

This work has been partially supported by the Italian Ministry of University and Research under grant PRIN-2007F9437X (project *ANONIMO*).

## References

1. Samarati, P.: Protecting Respondents' Identities in Microdata Release. IEEE Transactions on Knowledge and Data Engineering 13(6), 1010–1027 (2001)
2. LeFevre, K., DeWitt, D.J., Ramakrishnan, R.: Incognito: Efficient Full-domain $k$-Anonymity. In: Proc. of SIGMOD 2005, pp. 49–60. ACM Press, New York (2005)
3. Machanavajjhala, A., Gehrke, J., Kifer, D., Venkitasubramaniam, M.: $l$-Diversity: Privacy Beyond $k$-Anonymity. In: Proc. of ICDE 2006. IEEE Comp. Soc., Los Alamitos (2006)
4. Li, N., Li, T., Venkatasubramanian, S.: $t$-Closeness: Privacy Beyond $k$-Anonymity and $l$-Diversity. In: Proc. of ICDE 2007, pp. 106–115. IEEE Comp. Soc., Los Alamitos (2007)
5. Xiao, X., Tao, Y.: $m$-Invariance: Towards Privacy Preserving Re-publication of Dynamic Datasets. In: Proc. of SIGMOD 2007, pp. 689–700. ACM Press, New York (2007)
6. Byun, J.W., Sohn, Y., Bertino, E., Li, N.: Secure Anonymization for Incremental Datasets. In: Jonker, W., Petković, M. (eds.) SDM 2006. LNCS, vol. 4165, pp. 48–63. Springer, Heidelberg (2006)
7. Pei, J., Xu, J., Wang, Z., Wang, W., Wang, K.: Maintaining $k$-Anonymity against Incremental Updates. In: Proc. of SSDBM 2007. IEEE Comp. Soc., Los Alamitos (2007)
8. Fung, B.C.M., Wang, K., Fu, A.W.C., Pei, J.: Anonymity for Continuous Data Publishing. In: Proc. of EDBT 2008, pp. 264–275. ACM, New York (2008)

# A Bipartite Graph Framework for Summarizing High-Dimensional Binary, Categorical and Numeric Data

Guanhua Chen[1], Xiuli Ma[1,2,*], Dongqing Yang[1,3], Shiwei Tang[1,2], and Meng Shuai[2]

[1] School of Electronics Engineering and Computer Science,
Peking University, Beijing, 100871, China
[2] Key Laboratory of Machine Perception (Ministry of Education),
Peking University, Beijing, 100871, China
[3] Key Laboratory of High Confidence Software Technologies (Ministry of Education),
Peking University, Beijing, 100871, China
{ghchen,maxl,shuaimeng}@cis.pku.edu.cn, {dqyang,tsw}@pku.edu.cn

**Abstract.** Data summarization is an important data mining task which aims to find a compact description of a dataset. Emerging applications place special requirements to the data summarization techniques including the ability to find concise and informative summary from high dimensional data, the ability to deal with different types of attributes such as binary, categorical and numeric attributes, end-user comprehensibility of the summary, insensibility to noise and missing values and scalability with the data size and dimensionality. In this work, a general framework that satisfies all of these requirements is proposed to summarize high-dimensional data. We formulate this problem in a bipartite graph scheme, mapping objects (data records) and values of attributes into two disjoint groups of nodes of a graph, in which a set of representative objects is discovered as the summary of the original data. Further, the capability of representativeness is measured using the MDL principle, which helps to yield a highly intuitive summary with the most informative objects of the input data. While the problem of finding the optimal summary with minimal representation cost is computationally infeasible, an approximate optimal summary is achieved by a heuristic algorithm whose computation cost is quadratic to the size of data and linear to the dimensionality of data. In addition, several techniques are developed to improve both quality of the resultant summary and efficiency of the algorithm. A detailed study on both real and synthetic datasets shows the effectiveness and efficiency of our approach in summarizing high-dimensional datasets with binary, categorical and numeric attributes.

**Keywords:** Data Summarization, High-Dimensional Data, Bipartite Graph, the MDL Principle.

## 1 Introduction

Data summarization is an important data mining task which aims to find a compact description of a dataset. Many techniques have been proposed to summarizing

[*] Corresponding author.

transactional databases [19], categorical databases [5, 18], frequent patterns [2, 8], query results [11], texts [13], rules [9, 12] and graphs [14, 17]. Today, numerous systems and applications need to analyze high-dimensional datasets such as the gene expression data, collections of sensor network readings, document-term relationship data and market basket data, etc.. Below, we list some of these application domains.

**Gene Expression Data.** In biomedical research, high-throughput experimental data, like microarray, is a typical high-dimensional data, where the expression level of a set of genes or proteins is recorded under a set of conditions. The expression levels under each condition can be either binary (expressed or not) or categorical (high, medium, and low expression level) or numeric (the actual value of expression). Summarizing such data can yield groups of genes with common functionalities and behaviors, helping to find potential interactions among genes.

**Document Term Data.** In a search engine system, the document term relationship can be represented in a matrix, in which each row corresponds to a document and each column corresponds to a term. Each cell *ij* of the matrix can be either binary (the *j*th term is contained by the *i*th document) or numeric (the value of *tf/idf* ). Summarizing such data can generate groups of documents with similar semantics, which can be used to classify documents and improve the performance of the document index.

**Sensor Network Data.** Sensor network data contains periodic readings from a large set of sensor nodes over a period of time. If we regard sensors as objects and time points as the set of attributes, the groups of sensors with similarly evolving readings can be discovered, which can be used to direct the energy efficient data collection schemes such as online aggregation and semantic routing.

**Market Basket Data.** Market basket data contains information about products bought by customers. Regarding customers as rows and products as columns, the data elements can be binary (purchased or not) or numeric (quantity/sales of a purchase for a product). Summarizing the data to find groups of customers with similar purchasing patterns can be helpful to customer segmentation and targeted advertising.

A common topic in the above applications is the need to analyze datasets containing a large set of attributes which could be binary, categorical, numeric, or mixture of them. Summarizing such data is difficult, since it is hard to formulate the similarity between such objects in a uniform measure, due to the variant types of attributes. The requirements posed by such applications motivated us to develop a new solution that aims to summarize high-dimensional data with variant types of attributes effectively and efficiently.

## 1.1 Requirements for Techniques to Summarize High-Dimensional Data

In this section, we present the requirements to summarize high-dimensional data, which motivated our work on developing new solution to find high quality summary for high-dimensional data with variant types of attributes.

**Concise and informative summary.** A summary should be a compact description of a set of objects with little information loss. These two criteria are contradictive, for the coverage of a summary is positively correlated to its size in most of time. A good solution should be able to trade off between the conciseness and information loss, to construct the summary with reduced representation size while the important characteristics of the data are emphasized.

**Adaptive to variant attributes.** Datasets from different applications may contain variant types of attributes. Further, a dataset may contain attributes with different types. For example, the forest fires dataset [6] contains both categorical and numerical attributes. The adaptiveness to variant types of attributes is important for the usability of a data summarization technique.

**Comprehensibility of summary.** Data summarization aims at providing a comprehensible overview of data, thereby allowing an analyst to get an idea about the data easily without actually having to analyze the entire data. Thus, it is critical to design the summary representation which can be easily perceived by user. For high-dimensional datasets, it is especially important for a summary having a simple representation, because most visualization techniques do not work well for high-dimensional data.

**Scalability.** Finally, the solution should be efficient and scale well with the dimensionality and the size of input data.

Other requirements related to the usability issue include insensitiveness to outliers and the ability to handle missing values, which are common in real world datasets.

## 1.2 Contributions and Paper Layout

We propose a new research problem of discovering optimal summary of high-dimensional data with variant types of attributes. A general framework, referred as BIGFIRES (a $\underline{\text{BI}}$partite $\underline{\text{G}}$raph $\underline{\text{F}}$ramework with $\underline{\text{I}}$llustrative $\underline{\text{RE}}$presentatives for $\underline{\text{S}}$ummarization), is developed to satisfy all above requirements. BIGFIRES extracts the set of most representative objects as the summary of high-dimensional data, which is compact and of high coverage. The most representative objects are listed in the descending order of their coverage, along with their values in each attribute. Since each element of the summary is a real object from the data without any transformation, such a representation by illustration is easy to be understood by user. Both theoretical analysis and empirical evaluation shows that BIGFIRES scales quadratic to the number of objects, and linearly with the dimensionality of objects.

The rest of the paper is organized as follows. In Section 2, the problem of constructing summary by the set of most representative objects is defined and related concepts are introduced. In Section 3, we introduce our algorithm. In Section 4, we evaluate the effectiveness and efficiency of BIGFIRES on real and synthetic datasets. We introduce related work in Section 5 and then summarize our work and discuss some future research directions in Section 6.

## 2 Problem Statement

In this section, we state the problem of summarizing high-dimensional data sets that have binary, categorical and numeric attributes. We denote the dataset as $D = O \times A$, where $O$ stands for a set of objects $O= \{O_1, O_2, ..., O_n\}$ and $A$ stands for a set of attributes $A = \{A_1, A_2,..., A_m\}$, for each $O_i \in O$, we define $O_i = \{O_{i1}, O_{i2}, \dots , O_{im}\}$ $(1 \leq i \leq n)$, in which $O_{ij}$ is the value of the object $O_i$ on the attribute $A_j$.

To achieve a uniform and meaningful similarity measure for objects with different types of attributes, a bipartite graph representation is introduced to represent the data. We first present the bipartite graph representation for a dataset $D$.

### 2.1 The Bipartite Graph Representation

A bipartite graph $G=<V_1, V_2, E>$ is constructed to represent $D$ in two steps:

**Step 1. Preprocessing of numeric attributes.** For each numeric attribute $A_i \in A$, transform $A_i$ into a categorical attribute $A_i'$ by techniques such as discretization, then replace the original numeric values of $A_i$ with categories in $A_i'$. This is a preprocessing step in which better discretization quality could be achieved with advices from domain experts. For instance, an attribute of gene expression level is usually transformed into a categorical attribute with three categories: low, medium, and high expressed, each of which has its range of expression level. Data discretization is common in handling numeric attributes of high-dimensional data, such as subspace clustering [3] as well as data summarization [5]. After this step, the dataset contains only binary and categorical attributes.

**Step 2. Transforming *D* into *G*.** In this step, we construct a bipartite graph from a dataset with only binary and categorical attributes. The set of objects and the values of attributes are mapped into the two disjoint groups of nodes in the resultant bipartite graph, while each edge represents for an object containing a value. Without any extra effort to handle the missing values, they can be ignored safely in this transformation. The detailed transform process is listed as follows:

```
1  For each object $O_i \in O$, put $O_i$ into $V_1$
2    For each value of $O_i$ on attribute $A_j$ (i.e. $O_{ij}$)
3      If $A_j$ is a binary attribute
4        Put $A_j$ into $V_2$ if $A_j \notin V_2$
5        Put an edge $e<O_i, A_j>$ into $E$ if $O_{ij} = 1$
6      If $A_j$ is a categorical attribute
7        Put $A_j.O_{ij}$ into $V_2$ if $A_j.O_{ij} \notin V_2$
8        Put an edge $e<O_i, A_j.O_{ij}>$ into $E$
9    End For
10 End For
```

After the transformation, the bipartite graph $G$ contains all objects of $D$ in $V_1$ (i.e. $V_1 = O$), and all binary attributes and values of categorical attributes are contained in $V_2$, while $E$ contains relationships between objects and the values.

We define the value set of an object $O_i$ as its neighbor set in $G$, denoted as $N_i$.

**Exmaple 1.** A synthetic dataset of cars is present in table 1, which has its numeric attribute HorsePower discretized to a categorical attribute with equi-length interval. The constructed bipartite graph is shown in Figure 1. The value set of Car1 is $N_1$={BC.Silver, CD, Weight.100~200}.

**Table 1.** A synthetic dataset of cars

| Cars | Body Color (BC) | DVD | CD | Horsepower (HP) | … |
|---|---|---|---|---|---|
| Car1 | Silver | 0 | 1 | 100~200 | … |
| Car2 | Black | 1 | 0 | 200~300 | … |
| Car3 | Black | 0 | 0 | 0~100 | … |
| Car4 | Silver | 0 | 1 | 100~200 | … |
| … | … | … | … | … | … |

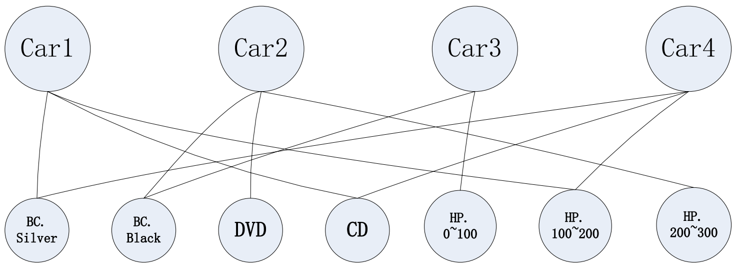


**Fig. 1.** An example bipartite graph constructed by the dataset in Table 1

## 2.2 The Problem Formulation with the MDL Principle

Minimum Description Length (MDL) principle [16] provides a direction for achieve a concise but informative summary from data. In the case of inferring a model from a dataset, a simple summary may result more description error than a complex summary, while the complex one may be too large, counteracting the reduction of description error it brings. The MDL principle states that the best model *M* to describe a set of data *D* is the one which minimizes the total representation cost of *M* and the data modeled by *M*, which can be denoted as cost($M$) + cost($D$ | $M$). From this perspective, we view summarization as an extraction process from *D* to a smaller set of representative objects *S* as the model to summarize the dataset *D* with an objective of minimize the sum of (i) the representation cost of *S* and (ii) the representation cost of *D* when it is represented by *S*.

Below, we define the concepts of summary and representation cost with the bipartite graph to formulate the summarization problem with the MDL principle.

**Definition 1. (Summary)** A summary *S* of a dataset *D*, is a set of representative objects $S$ = {$O_{S1}$, $O_{S2}$, ..., $O_{Sl}$ } such that (i) $S \subseteq O$ and (ii) each $O_{Si} \in S$ represents a subset of *O*.

**Definition 2. (Representation cost of object(s))** The representation cost of an object $O_i$ is the size of its value set, that is, $cost(O_i) = |N_i|$. Further, the representation cost of a set of objects $P \subseteq O$ is the sum of the representation cost of the objects belonging to it, that is $cost(P) = \sum_{O_i \in P} cost(O_i)$. We can conclude that $cost(O) = \sum_{O_i \in O} |N_i| = |E|$.

**Definition 3. (Representation cost for an object $O_i$ represented by another object $O_j$)** For two objects $O_i$, $O_j \in O$, the representation cost of $O_i$ represented by $O_j$ is the edit distance of the value sets of the two nodes. $cost(O_i \mid O_j) = |N_i - N_j| + |N_j - N_i|$.

The intuition of the concept of representation cost between objects is that an object $O_i$ can be transformed to $O_j$ by adding into $N_i$ the values that are not in $N_i$ but in $N_j$ (i.e. $N_j - N_i$), then removing the values not in $N_j$ but in $N_i$ (i.e. $N_i - N_j$) from $N_i$. In addition, we have $cost(O_i \mid O_j) = cost(O_j \mid O_i)$ and $cost(O_i \mid O_i) = 0$.

**Definition 4. (Representation cost for an object $O_i$ represented by a summary $S$)** For an object $O_i \in O$ and a summary $S$, the representation cost of $O_j$ represented by $S$ is

$$cost(O_i \mid S) = \begin{cases} \min_{O_{sj} \in S} cost(O_i \mid O_{sj}), \text{ if } \min_{O_{sj} \in S} cost(O_i \mid O_{sj}) < cost(O_i) \\ cost(O_i), \text{ otherwise} \end{cases}$$

When the representation cost for $O_i$ represented by one of the object $O_{Sl} \in S$ achieves a minimal cost $c$, we compare $c$ with $cost(O_i)$. If $c$ is less than $cost(O_i)$, which means the representation cost of $O_i$ can be smaller when represented by $S$, then the representation cost is the minimum $c$. Otherwise, when $c$ is not less than $cost(O_i)$, since $S$ cannot reduce the representation cost for $O_i$, the cost of $O_i$ remains without any change.

**Definition 5. (Representation cost for a set of objects $O$ represented by summary $S$)** For a set of objects $O$ and a summary $S$ of $O$, the representation cost of $O$ represented by $S$ is $cost(O \mid S) = \sum_{O_i \in O} cost(O_i \mid S)$.

**Example 2.** As shown in Figure 2(a), a bipartite graph $G$ is presented with the set of objects $\{O_1, O_2, O_3, O_4\}$ and the set of values $\{v_1, v_2, v_3, v_4, v_5\}$. $cost(O_1) = 3$, for $N_1 = \{v_1, v_2, v_3\}$; $cost(O_2) = 4$, for $N_2 = \{v_1, v_2, v_3, v_4\}$; $cost(O_3) = 2$, for $N_3 = \{v_4, v_5\}$, $cost(O_4) = 3$, for $N_4 = \{v_3, v_4, v_5\}$; $cost(O) = 12$. $cost(O_2 \mid O_1) = 1$, for there is only one value should be edited (remove $v_4$ from $N_2$) when transforming $N_2$ to $N_1$.

So far, we have defined the concepts of summary and representation cost in several cases. Below, we define the best summary which achieves the minimal total representation cost that follows the MDL principle.

**Definition 6. (Optimal Summary)** A summary $S$ of a set of objects $O$ following the MDL principle is referred as the optimal summary, if the total representation cost of summary $S$ and $O$ represented by $S$ is minimized. Formally, the optimal summary $S$ of $O$ satisfies:

$$\operatorname{argmin}_{S \subseteq O} (cost(S) + cost(O \mid S)).$$

Using the Definition 6, the problem of summarizing a high dimensional dataset is reducing to discovering the optimal summary of the bipartite graph that represents the dataset. In this paper, we investigate efficient methods to discover the optimal summary.

**Example 3.** The optimal summary of the graph in Figure 2(a) is $S = \{O_1, O_3\}$, where $cost(S) + cost(O \mid S) = cost(O_1) + \text{cost}(O_3) + cost(O_1|O_1) + cost(O_2|O_1) + cost(O_3|O_3) + cost(O_4|O_3) = 3 + 2 + 0 + 1 + 0 + 1 = 7$. The summary $S$ is presented in Figure 2(b) as graph $G'$.

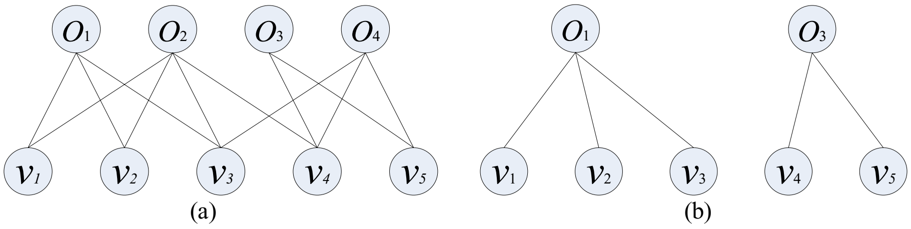


**Fig. 2.** An Example Bipartite Graph (a) graph $G$; (b) graph $G'$, the summary of $G$

## 3 Discovering of the Optimal Summary

The problem of discovering the optimal summary for a given set of objects is difficult. To achieve the exact answer, one has to enumerate all subset of $O$ and pick the one with the minimal total representation cost. The complexity of such a brute force solution is $O(2^{|O|})$, which is computational infeasible.

A heuristic solution for the optimal summary problem is to find a set of most representative objects to approximate the exact optimal summary. We first formulate the measure of representativeness of an object.

Intuitively, an object with high representativeness can represent many other objects with a significant reduction in the total representation cost. We use the total reduced cost of an object to measure its representativeness. First, we define the reduced cost between objects.

**Definition 7. (The reduced cost of object $O_i$ represented by $O_j$)** The reduced cost of object $O_i$ represented by $O_j$ is defined as $RC(O_i \mid O_j) = cost(O_i) - cost(O_i \mid O_j)$

The reduced cost of object $O_i$ when represented by $O_j$ is used to measure how well $O_i$ can be represented by $O_j$: the larger reduced cost, the better $O_j$ can represent $O_i$. On the other hand, if $RC(O_i \mid O_j) \leq 0$, we can conclude that it is infeasible for $O_j$ to represent $O_i$.

For an object $O_j$ to represent $O_i$ well, a positive reduced cost is not sufficient. Figure 3 shows an example: $RC(v \mid u) > 0$, but $u$ cannot represent $v$ well. To tackle this problem, we use the relationship of valid representation between objects $O_i$ and $O_j$.

**Definition 8. (The relationship of valid representation between objects $O_i$ and $O_j$)** Objects $O_i$ can be a valid representation of $O_j$ if and only if $RC(O_i \mid O_j) > 0$ and $RC(O_j \mid O_i) > 0$. The relationship of valid representation between $O_i$ and $O_j$ is denoted as

$r(O_i, O_j) = \begin{cases} true, \text{if } RC(O_i|O_j) > 0 \text{ } and \text{ } RC(O_j|O_i) > 0 \\ false, \text{ otherwise} \end{cases}$. In fact, it is straight forward to have $r(O_i, O_j) = r(O_j, O_i)$.

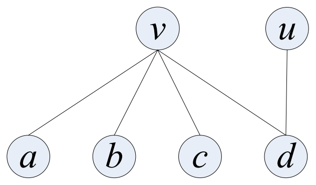


**Fig. 3.** $RC(v \mid u) > 0$, but $u$ cannot represent $v$ well.

The relationship of valid representation is of several desirable properties.

**Property 1.** If $r(O_i, O_j)=true$, $|N_i \cap N_j| > max(|N_i|/2, |N_j|/2)$.

**Proof**. From $RC(O_i \mid O_j) > 0$ and $RC(O_j \mid O_i) > 0$,

we have $|N_i| - (|N_i - N_j| + |N_j - N_i|) > 0$ … (1)

and $|N_j| - (|N_j - N_i| + |N_i - N_j|) > 0$ … (2)

From (1) $|N_i| - (|N_i - N_j| + |N_j - N_i|)$

$= |N_i| - (|N_i - (N_i \cap N_j)| + |N_j - (N_i \cap N_j)|)$

$= |N_i| - (|N_i| + |N_j| - 2|N_i \cap N_j|) = 2|N_i \cap N_j| - |N_j| > 0$

$|N_i \cap N_j| > |N_j|/2$ ….(3)

Similarly, From (2) $|N_i \cap N_j| > |N_i|/2$……(4)

Together with (3) and (4) $|N_i \cap N_j| > max(|N_i|/2, |N_j|/2)$. ∎

This property means when two objects $O_i$ and $O_j$ can be valid representation to each other, the number of common values is greater than half number of their values.

**Property 2.** If $r(O_i, O_j)=true$, then $|N_j|/2 < |N_i| < 2 \cdot |N_j|$

**Proof**. From $RC(O_i \mid O_j) > 0$ and $RC(O_j \mid O_i) > 0$,

From equation (3) and (4) in the above proof, we have

$2|N_i \cap N_j| > |N_i|$ and $2|N_i \cap N_j| > |N_j|$,

From the former we have $|N_i| < 2|N_i \cap N_j| \leq 2 \cdot |N_j|$,

Similarly, from the latter we have $|N_j|/2 < |N_i \cap N_j| \leq |N_i|$,

In summary, we have $|N_j|/2 < |N_i| < 2 \cdot |N_j|$. ∎

If $O_i$ is a valid representation of $O_j$, this property limits the size of $N_i$ by no less than half and no more than twice the size of $N_j$. In other words, an object should have similar size on the value set with that of objects it represents.

We denote the set of objects can be validly represented by object $O_i$ as $VR_i$, formally, $VR_i = \{ O_j \mid r(O_i, O_j) = true, O_j \in O\}$ . We call $VR_i$ as the validly represented set of object $O_i$.

**Definition 9.** The total reduced cost of object $O_i$ is the sum of the reduced cost of objects in $VR_i$ when represented by $O_i$. Formally, $RC(VR_i \mid O_i) = \sum_{O_j \in VR_i} RC(O_j | O_i)$

The total reduced cost of an object is a good measure for selecting the most representative objects which compose an approximate optimal summary. In the rest of this section, we will introduce our summarization method based on greedy search, as well as the optimization heuristics.

### 3.1 The Greedy Algorithm to Discover Approximate Optimal Summary

**Algorithm: Approximate Optimal Summary**
INPUT: A Bipartite Graph $G<V_1, V_2, E>$
OUTPUT: Summary $S$

```
1  Initialization: for each object $O_i$ in $V_1$, calculate $VR_i$ and $RC(VR_i \mid O_i)$
2  $O_{remain} = V_1$, $S=\emptyset$
3  While ($O_{remain} \neq \emptyset$)
4      Find $O_i$ with maximal $RC(VR_i \mid O_i)$ in $O_{remain}$
5      $S = S \cup \{ O_i \}$
6      $O_{remain} = O_{remain} - VR_i$
7      For each object $O_j$ in $O_{remain}$
8          $A = VR_i \cap VR_j$
9          If ($A \neq \emptyset$)
10             Update $VR_j = VR_j - A$
11             Update $RC(VR_j \mid O_j) = RC(VR_j \mid O_j) - \sum_{O_k \in A} RC(O_k \mid O_j)$
12     End for
13 End while
14 Output $S$.
```

The algorithm is designed to discover approximate optimal summary using a greedy search approach. At the initialization phase, for each object, the validly represented set is calculated, along with the total reduced cost. In the following iterations, it repeats Lines 3-12 by selecting the object $O_i$ which has the maximal total reduced cost (Line 4) and adding it into the summary (Line 5). After removing the objects that are validly represented by $O_i$ from the remaining object set $O_{remain}$ (Line 6), it is necessary to update the validly represented set and the total reduced cost for each remaining object, if the validly represented set contains any object that has just been removed (Line 7-12). The iteration procedure is terminated when the remaining object set is empty, that is, all objects have been covered by the summary.

The process of initialization calculates the reduced cost of each pair of objects, in which the cost of computation for the overlap of two sets of values is linear to the average size of the value set, thus takes $O(n^2d)$ in time, where $n$ is the number of objects and $d$ is the dimensionality. The dimensionality is the sum of the number of categorical/numeric attributes and the average transaction length (count of '1') for binary attributes. The process of finding the object with maximal total reduced cost is $O(n)$, and updating the total reduced cost for each of the remaining objects is also in time $O(n)$. The number of iterations is $n$ in worst case, so it can be terminated in time $O(n^2)$. Hence, the cost of the algorithm is $O(n^2d + n^2)$.

### 3.2 Optimization Heuristics

We developed two optimization heuristics to improve the effectiveness and efficiency of the algorithm.

### 3.2.1 Outlier Detection

An outlier is defined as an object that has a small set of validly represented objects. For instance, an object which can only be validly represented by itself could not contribute to reduce the total representation cost.

In the algorithm, each outlier poses an iteration and being added into the summary. Detecting outlier early benefits both the performance of the algorithm and the conciseness of the resultant summary. There are two kinds of outliers which need to be handled respectively.

**Natural Outliers:** The natural outliers are the set of objects each of which can only be validly represented by itself. Such kind of outliers is caused by noise or exception. We can detect natural outliers in the initialization process: when the computation of an object's validly represented set is over, we can check its size to identify whether the object is an outlier. If so, it should be removed from the set of objects. However, natural outliers may be valuable for carrying exceptional patterns. They also can be recorded for applications that are interested in such patterns.

**Derived Outliers:** The derived outliers are generated in the update phase of the algorithm, where the set of valid represented objects is reduced to a small one. Such outliers can be detected right after the update phase and removed from the set of remaining objects.

The ability of handling these two kinds of outliers is helpful to discover a more concise summary.

### 3.2.2 Reassignment of Objects

The reassignment is a post-processing step in which the validly represented set of each object in summary is turned into the best represented set. The best represented set is defined as follows: for an object $O_i$ in summary, its best represented set contains those objects, each of which achieves the maximal reduced cost with $O_i$. Formally, for an object $O_i \in S$, $BR_i$ is the best represented set of $O_i$, where $BR_i = \{O_j \in O \mid \forall O_k \in S, cost(O_j \mid O_i) \geq cost(O_j \mid O_k)\}$. In other words, objects in $O_i$'s best represented set are best represented by $O_i$. Instead of the validly represented set, the best represented set is used to calculate the coverage and accuracy (see Section 3.4) of each object in the summary, which are meaningful measures to understand the characteristic of each objects in the summary.

The process of reassignment is similar to that of point assignment in K-means: each point is assigned to its nearest center point. Natural outliers are excluded because there is not any object in the summary that can be a valid representation of them. The reassignment needs one pass through all objects. For each object $O_i$, it searches in the summary for an object $O_j$ that has the least representation cost when $O_i$ is represented by $O_j$. When the object $O_j$ is found, the object $O_i$ is assigned into $BR_j$. The cost of reassignment is $O(kn)$, where $k$ is the size of the summary and $n$ is the number of objects.

## 3.3 Top-*k* Summary

The MDL principle provides an automatic approach to find the optimal summary for the given dataset, where the representative objects are selected to minimize the total representation cost. When the summary following the MDL principle (say, MDL-summary) is still too large to study by users one by one, a refined summary with limited size and most representative objects (say, top-*k* summary) is preferred.

Based on BIGFIRES, it is straightforward that a refined summary can be achieved by applying a top-*k* query on the MDL-summary. For the objects are selected in the descending order of their representativeness, the top-*k* representative objects are the first *k* objects in the MDL-summary. That is, a top-*k* summary is a part of an MDL summary with the first *k* objects.

### 3.4 Quality Evaluation

One way to evaluate the quality of the summary is to calculate the accuracy of each value of each object in summary, which is defined as the percentage of objects in the best represented set $BR_i$ having the same value with $O_i$. Formally, given an object $O_i \in S$, for each value $O_{ij}$ of $O_i$, the accuracy is defined as

$$acc(O_{ij}) = |\{O_k| \ O_k \in BR_i \text{ and } O_{kj} = O_{ij}\}| \ / \ |BR_i|.$$

Further, the average accuracy for an object $O_i$ in summary is defined as

$$average\_acc\ (O_i) = \sum acc(O_{ij})/|N_i|$$

To measure the representativeness of an object in a summary, the coverage of an object $O_i \in S$ is defined as the percentage of objects that can be best represented by it. Formally, given an object $O_i \in S$, the coverage of $O_i$ is denoted as

$$cover(O_i) = |BR_i| \ / \ |O|.$$

Further, to measure the representativeness of a summary $S$, the coverage of $S$ is defined as

$$cover(S)=| \cup_{Oi \in S} BR_i | \ / \ |O|.$$

## 4 Experiments

In this section, we present the experimental results to evaluate the effectiveness and efficiency of BIGFIRES on a variety of real and synthetic datasets. The algorithm of BIGFIRES is implemented in Java. We first import the datasets into a bipartite graph, which contains two sets of nodes $V_1$ and $V_2$, $V_1$ stands for the set of objects in the dataset while $V_2$ includes the binary attributes, the categories for categorical attributes and discrete intervals for numeric attributes. We stored the set of edges by recording the set of neighbors (nodes from $V_2$) of each node in $V_1$. The set of neighbors are stored in a hash map by which we can calculate the overlap of neighbors between nodes efficiently in the manner of hash join. All experiments were run on a 2.0GHz AMD Sempron machine running Windows XP, and equipped with 1.5GB RAM. We set the JVM heap space to 512MB in all experiments.

## 4.1 Experimental Datasets

In this section, we describe the datasets used in our empirical evaluation. We use three real datasets and two synthetic datasets to explore the effectiveness. The datasets are listed in Table 2.

**Table 2.** Experimental datasets

| **Name** | **Attributes** | $\lvert V_1 \rvert$ | $\lvert V_2 \rvert$ | **Edge size** | **Type** | **Missing Value** |
|---|---|---|---|---|---|---|
| Mushroom | 19 | 8,124 | 108 | 151,876 | Categorical | Yes |
| Forest Fire | 10 | 517 | 88 | 5,170 | Categorical, Numeric | No |
| Drug Order | 137 | 328 | 137 | 13,929 | Binary | No |
| T10I4D100K | ~1K | 100K | ~1K | ~1000K | Binary | No |
| T40I10D100K | ~1K | 100K | ~1K | ~4000K | Binary | No |

The following three real datasets are mainly used to evaluate the effectiveness of BIGFIRES, while the two synthetic datasets are used to evaluate the efficiency.

**Mushroom Dataset.** It is a classical dataset with two forms: one is from UCI Machine Learning Repository [4] which is of 8,124 instances and 22 categorical attributes plus one classification attribute (edible mushrooms or poisonous mushrooms) used for classification; the other one is transformed to the transactional dataset used for frequent itemset mining in FIMI Repository [1]. Both forms can be imported into the same bipartite graph structure, in which $V_1$ is the set of 8,124 instances of mushrooms and $V_2$ is the set of 119 items/categories. For ease of explaining the results, we use the UCI form with categorical attributes. 4 attributes are deleted from the dataset beforehand for they are almost constant attributes and contribute little for classification. After the deletion, there are 18 categorical attributes remaining with one classification attribute, which turns to be 108 items/categories in total. There are over 1.6% missing values in the dataset, occurring in the 11th attribute (stalk-root).

**Forest Fire Dataset.** This dataset is also from UCI Machine Learning Repository. It contains 517 instances of forest fire and 13 attributes, in which 4 attributes (Location X/Y, Month and Day) are categorical and the other 9 attributes (FFMC [18.7 to 96.20], DMC [1.1 to 291.3], DC [7.9 to 860.6], ISI [0.0 to 56.10], temperature [2.2 to 33.30], relative humidity [15.0 to 100], wind [0.40 to 9.40], rain [0.0 to 6.4] and area [0.00 to 1090.84]) are numeric. This dataset is used for regression task, where the aim is to predict the burned area of forest fires, in the northeast region of Portugal. More information of this dataset refers to [6]. Unlike its original task, we use this data for discovering the characteristic of these forest fires. Similar to the Mushroom data, we removed two attributes from the dataset: rain and area whose values are mostly 0. In additional, the location spatial coordinate X and Y are combined into one categorical attribute to avoid being processed separately. After the transformation, there are 10 attributes remaining and 5,170 edges in total for there are no missing values (each instance has 10 edges).

**Drug Order Dataset.** This dataset is a real order dataset with 328 distributors ordering 137 kinds of drugs from a drug factory. We use it as a binary dataset: $V_1$ represents for distributors and $V_2$ represents for the set of different drugs. Each edge between one distributor and one drug in the bipartite graph means there is at least one order that the distributor had purchased the drug. Summarization on such data is helpful to discover some typical purchase patterns, while providing knowledge for customer segmentation.

**Synthetic Datasets.** Other than the real datasets, synthetic datasets are used to evaluate the efficiency of our method. We select two synthetic datasets T10I4D100K andT40I10D100K as binary datasets. Although the two datasets are of same number of nodes, the latter's number of edges (~4000K) is four times than that of the former (~1000K), fragments of different size from these two dataset are used to test the scalability of our method.

## 4.2 Effectiveness Evaluation

We first evaluate the effectiveness of our summarization method on three real datasets Mushroom, Forest Fire and Drug Order respectively.

For limit of space, we couldn't list all representative objects with the full set of attributes. We select attributes with high accuracy values to present the top-*k* summary. Table 3 is an overview of the summaries discovered by BIGFIRES from the three real datasets. The summaries are relative small to the original object set, while cover most objects in the data. The following sub-sections get deeper into the results of these three dataset respectively.

**Table 3.** An overview of the set of representative objects from real datasets

| **Dataset** | $\lvert S\rvert$ | $\lvert S\rvert / \lvert O\rvert$ | ***cover*(*S*)** |
|---|---|---|---|
| Mushroom | 16 | 0.002 | 100% |
| Forest Fire | 30 | 0.058 | 97.7% |
| Drug Order | 24 | 0.073 | 74.1% |

**Summary of Mushroom Dataset**

In this experiment, we are interested in the correlation between the classification attribute (edible or poisonous) and other attributes. As shown in Table 3, the resultant set of representative objects composes 16 representative mushrooms, which can represent all 8124 mushrooms. From the summary, there are 13 representative mushrooms get very high support on the classification attribute (over 95%). In Table 4 we list the top-5 representative mushrooms with 6 selected attributes, which cover over 75% mushrooms in total. The values of each representative mushrooms are presented with their accuracy. Thus, the characteristic of each type of mushroom can be perceived by user easily.

**Table 4.** Top-5 representative Mushrooms, attribute are chosen by high accuracy values

| $O_i$ | e /p | bruise | odor | gill spacing | gill size | stalk shape | ring type | cover ($O_i$) |
|---|---|---|---|---|---|---|---|---|
| 2406 | e 96.7% | t 99.5% | n 74.5% | c 96% | b 95.3% | t 74.0% | p 100% | 30.5% |
| 3855 | p 100% | f 100% | f 100% | c 100% | b 100% | e 100% | l 100% | 15.8% |
| 6228 | p 100% | f 100% | f 41.9% | c 100% | n 98.9% | t 98.9% | e 98.9% | 12.1% |
| 1320 | e 100% | f 98.3% | n 98.3% | w 98.3% | b 100% | t 95.4% | e 95.4% | 9.9% |
| 6692 | p 100% | f 100% | s 44.4% | c 100% | n 100% | t 100% | e 100% | 9.2% |

Obviously, each representative mushroom gets high accuracy on the classification attribute, which reveals that there are strong correlations between the classification attributes and the other attributes. For example, in Table 4, we can find that most edible mushrooms are of no odor (odor is ‘n’), while those poisonous are possibly of fishy or spicy odor (odor is ‘f’ or ‘s’). Furthermore, a mushroom with broad gill (gill size is ‘b’) and tapering stalk (stalk shape is ‘t’) is probably a edible one. The summary with representative objects provides great conveniences for such kind of analysis.

**Summary of Forest Fire Dataset**

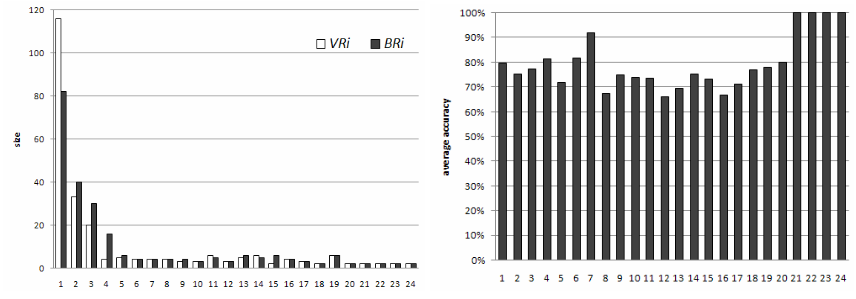
In this experiment, we evaluate the effectiveness of BIGFIRES on the Forest Fire dataset with 517 instances and 10 attributes (including 3 categorical and 7 numeric attributes). We use an equi-length intervals approach to discretize the values of the 7 numeric attributes. The number of intervals is set to 5. There are 30 representative instances returned. Table 5 is the top-5 representative forest fire which covers over 50% instances. When study on the result, we can discover a strong correlation between the forest fire with seasons (mostly September and August) through the high accuracy with the values on Month. Furthermore, high FFMC (80.7 - 96.2) and low ISI (0.0 - 11.2) are common in most forest fires.

**Summary of Drug Order Dataset**

The drug order dataset is a transactional dataset which can be considered as a binary dataset by transforming it into a binary matrix. In the binary matrix, the rows stand for 328 drug distributors and the columns stand for 137 kinds of drugs. The summary composes of 24 representative distributors. The first representative distributor reveals a huge pattern, for it can valid represent 116 distributors (35% of all distributors), and best represent 82 distributors (25% of all distributors). The representative distributor of the huge pattern has ordered 97 kinds of drugs, which can be regarded as an approximate frequent pattern containing 97 items with a support of 25%, and the approximation ratio is nearly 80% (see Figure 4(b)).

**Table 5.** Top-5 representative Forest Fires, attributes are chosen by high accuracy values

| $O_i$ | Month | FFMC | DMC | DC | ISI | cover ($O_i$) |
|---|---|---|---|---|---|---|
| 353 | Sep 91.8% | 80.7 - 96.2 100% | 59.1 -117.2 76.3% | 690.1 - 860.6 90.72% | 0.0 - 11.2 90.72% | 18.8% |
| 94 | Aug 87.7% | 80.7 - 96.2 100% | 117.2 - 175.2 53.9% | 519.5 - 690.1 90.77% | 0.0 - 11.2 81.54% | 12.6% |
| 119 | Mar 84.3% | 80.7 - 96.2 96.1% | 1.1 - 59.1 100% | 7.9 - 178.4 100.00% | 0.0 - 11.2 96.08% | 9.9% |
| 15 | Sep 100% | 80.7 - 96.2 100% | 117.2 - 175.2 80.0% | 690.1 - 860.6 94.29% | 0.0 - 11.2 94.29% | 6.8% |
| 268 | Aug 93.1% | 80.7 - 96.2 100% | 117.2 - 175.2 69.0% | 519.5 - 690.1 96.55% | 11.2 - 22.4 79.31% | 5.6% |



(a) (b)

**Fig. 4.** (a) the size of validly and best represented objects and (b) the average accuracy on the attributes of representative objects on the Drug Order dataset

As shown in Figure 4, we compare the size of validly represented set with the size of best represented set. The first four representative distributors in the Figure 4(a) can be regarded as four approximate frequent patterns with a support of 5%. These patterns are composed of 36 - 97 kinds of different drugs, which are too long to be discovered efficiently for frequent pattern mining algorithms [20]. From the Figure 4(b) we can see that the average accuracy of most representative distributors is over 70%. For the sake of privacy, details of the distributors and drugs are neglected.

## 4.3 Efficiency Experiment

This section evaluates the efficiency of BIGFIRES on both Mushroom and synthetic dataset by taking fragments of different size from them.

We first test the efficiency on the Mushroom dataset from first 1,000 instances to 8,000 instances (there are 8,124 instances in Mushroom dataset), comparing the cases with and without the procedure of reassignment. As shown in Figure 5(a), the performance is dominated by the initial phase of validly represented set computation for each instance with a cost of $O(n^2d)$, while the procedure of reassignment with cost of $O(kn)$ only add a relatively small times to the total execution process.

The two synthetic datasets, T10I4D100K and T40I10D100K, are used to test the scalability on both the size and dimensionality of instances. When the size of instances are same, the ratio of the size of edges of the two datasets is 1:4 approximately. As shown in Figure 5(b), similar to that of Mushroom dataset, the execution time is quadratic to the number of instance, while the time cost on T40I10D100K is 4 times of that of T10I4D100K, from which we can conclude that the execution time grows linear with the dimensionality of the dataset.

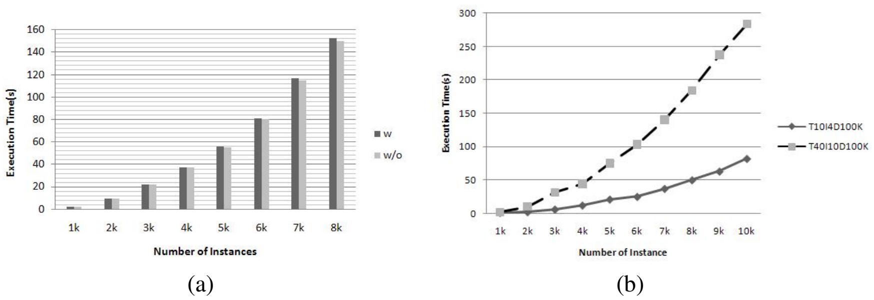


**Fig. 5.** (a) Execution Time on Mushroom dataset with and without reassignment. (b) Execution Time on synthetic datasets of T10I4D100K and T40I10D100K.

## 5 Related Work

Our work of data summarization is generally in the line of descriptive data mining. Clustering [3,7] is used to summarize numeric datasets. Agrawal. *et al* [3] propose a subspace clustering for high-dimensional datasets with a rectangle representation. Gao. *et al.* [7] extend descriptive data mining from a clustering description to a discriminative setting using a rectangle notation. Clustering techniques are not efficient for high-dimensional datasets and often generate a too large set of subspace clusters to be explored by users. Data summarization techniques on categorical and transactional databases are closely related with our work. Wang and Karypis [18] propose to summarize categorical databases by mining summary set. Each summary set contains a set of summary itemsets. A summary itemset is the longest frequent itemsets supported by a transaction. Chandola and Kumar [5] compress datasets of transactions with categorical attributes into informative representations by summarizing transactions. They show their methods are effective in summarizing network traffic. Siebes. *et al.* [15] propose to recognize significant itemsets by their ability to compress a database based on the MDL principles. The compression strategy can be explained as covering the entire database using the non-overlapped hyper-rectangles with no false positives allowed. Xiang. *et al.* [19] summarize transactional database with rectangle representation. They make use of the frequent itemsets to accelerate the process of generating the rectangles covering the dataset. Most of these methods are based on rectangle representation, while BIGFIRES is based on the most representative objects. The other difference is that these methods are not capable to deal with variant attributes in the data.

Another related topic is graph summarization. Navlakha. *et al.* [14] propose a graph summarization technique along with a MDL representation of the summary. Tian. *et al.* [17] propose a graph summarization approach by aggregating nodes with similar values and relationships. Although BIGFIRES works on a graph context, the goals are different: they try to find a small and high-level graph as the summary to exhibit the relationships between nodes, while we try to find the most typical nodes in the graph to represent objects in the dataset.

Compressing large Boolean matrices or transactional databases is becoming an increasingly important research topic as the size of databases is growing at a very fast pace. For instance, in [10], Johnson *et al.* tries to reorder the rows and columns so that the consecutive 1's and 0's can be com-pressed together. Summarization, on the other hand, aims at providing an overview of the data, thereby allowing an analyst to get an idea about the data without actually having to analyze the entire data.

In summary, we believe the above algorithms are orthogonal to ours in that we are more focused on the problem of choosing most representative objects in high-dimensional datasets with variant attributes.

## 6 Conclusion and Future Work

Data summarization is an important task of descriptive data mining. In this paper, we proposed the research problem of summarizing high-dimensional data with variant types of attributes. The generated summary is composed of most representative objects in the data. We formulate this problem in a bipartite graph scheme, in which the MDL principle is applied to characterize the task as a problem of discovering the optimal summary of the input data. A framework, BIGFIRES, is proposed with several optimization heuristics. The effectiveness and efficiency of our method is presented by experiments on several real and synthetic datasets.

In our future work, we plan to address two issues. First, the efficiency may be optimized further, especially improving the scalability by decreasing the cost to linear or sub-linear to the data size. Second, we plan to extend BIGFIRES with a hierarchical summary structure, which provides summary with varying detail levels and may be used as an index for similarity search in the high-dimensional databases.

**Acknowledgments.** We thank George Kollios and Jignesh Patel for their shepherding of this paper. We also thank Marianne Winslett and the reviewers of this paper for their constructive comments on a previous version of this manuscript.

## References

1. Frequent Itemset Mining Implementations Repository, `http://fimi.cs.helsinki.fi/`
2. Afrati, F., Gionis, A., Mannila, H.: Approximating a collection of frequent sets. In: Proc. KDD 2004 (2004)
3. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications. In: Proc SIGMOD 1998 (1998)

4. Asuncion, A., Newman, D.J.: UCI Machine Learning Repository. University of California, School of Information and Computer Science, Irvine, CA (2007), http://www.ics.uci.edu/~mlearn/MLRepository.html
5. Chandola, V., Kumar, V.: Summarization - Compressing data into an informative representation. Knowl. Inf. Syst. 12(3) (2007)
6. Cortez, P., Morais, A.: A Data Mining Approach to Predict Forest Fires using Meteorological Data. In: Proc. EPIA 2007 (2007)
7. Gao, B.J., Ester, M.: Turning Clusters into Patterns: Rectangle-based Discriminative Data Description. In: Proc. ICDM 2006 (2006)
8. Han, J., Wang, J., Lu, Y., Tzvetkov, P.: Mining top-k frequent closed patterns without minimum support. In: Proc. ICDM 2002 (2002)
9. Hu, M., Liu, B.: Mining and summarizing customer reviews. In: Proc. KDD 2004 (2004)
10. Johnson, D., Krishnan, S., Chhugani, J., Kumar, S., Venkatasubramanian, S.: Compressing large boolean matrices using reordering techniques. In: Proc. VLDB 2004 (2004)
11. Lakshmanan, L.V.S., Ng, R.T., Wang, C.X., Zhou, X., Johnson, T.J.: The Generalized MDL approach for Summarization. In: Proc. VLDB 2002 (2002)
12. Liu, B., Hu, M., Hsu, W.: Multi-level organization and summarization of the discovered rules. In: Proc. KDD 2000 (2000)
13. Mani, I.: Advances in Automatic Text Summarization. MIT Press, Cambridge (1999)
14. Navlakha, S., Rastogi, R., Shrivastava, N.: Graph Summarization with Bounded Error. In: Proc. SIGMOD 2008 (2008)
15. Siebes, A., Vreeken, J., Leeuwen, M.: Item Sets that Compress. In: Proc. SDM (2006)
16. Rissanen, J.: Modeling by the shortest data description. Automatica 14, 465–471 (1978)
17. Tian, Y., Hankins, R.A., Patel, J.M.: Efficient Aggregation for Graph Summarization. In: Proc. SIGMOD 2008 (2008)
18. Wang, J., Karypis, G.: On Efficiently Summarizing Categorical Databases. Knowl. Inf. Syst. 9(1), 19–37 (2006)
19. Xiang, Y., Jin, R., Fuhry, D., Dragan, F.F.: Succinct Summarization of Transactional Databases: An Overlapped Hyperrectangle Scheme. In: Proc. KDD (2008)
20. Zhu, F., Yan, X., Han, J., Yu, P.S., Cheng, H.: Mining Colossal Frequent Patterns by Core Pattern Fusion. In: Proc. ICDE 2007 (2007)

# Region Extraction and Verification for Spatial and Spatio-temporal Databases

Mark McKenney

Texas State University, Department of Computer Science
mckenney@txstate.edu

**Abstract.** Newer spatial technologies, such as spatio-temporal databases, geo-sensor networks, and other remote sensing methods, require mechanisms to efficiently process spatial data and identify (and in some cases fix) data items that do not conform to rigorously defined spatial data type definitions. In this paper, we propose an $O(n \lg n)$ time complexity algorithm that examines a spatial configuration, eliminates any portions of the configuration that violate the definition of spatial regions, and constructs a valid region out of the remaining configuration.

## 1 Introduction

Spatial databases have become rather mature and many commercial products are based upon them; however, newer technologies have introduced new problems in spatial data management, and highlighted problems in existing technologies that have yet to be fully addressed. We illustrate this with an example from moving objects databases. Current spatial technologies are built upon a rigid definition of spatial data types. For example, a *complex region* [1] (which we will refer to simply as a region) can consist of multiple faces, each containing zero or more holes (e.g., Italy has multiple islands as faces and a hole where Vatican City lies). A *moving region* is a complex region that moves and changes over time (Figure 1a). A well known operation over moving regions is to extract a region at a specific time $t$ [2]. Figure 1b depicts the region in Figure 1a at time $t$; note that the region contains some lines that do not form a face, but are one-dimensional components. Such components violate the definition of regions; yet, in the case of moving regions they are typically required to indicate that a face of a region is about to come into existence, or has just ceased to exist. Because the configuration at time $t$ contains such anomalies, we cannot simply extract the spatial configuration at that time and apply other spatial operations to it since spatial operations must have valid input to preserve type closure properties. Therefore, a mechanism is required to differentiate the invalid and valid portions of the configuration. In the case of the region in Figure 1b, we must be able to generate the region in Figure 1c. Additional occurrences of this type of problem arise with respect to data quality issues in fields such as geo-sensor networks and remote sensing, in which large amounts of data are generated.

More formally, we can characterize the problem highlighted above as follows: given a set of straight line segments $S$ in two dimensional space, does there exist

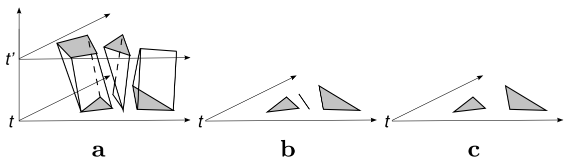


**Fig. 1.** A moving region (a), the scene at time $t$ (b), and the region at time $t$ (c)

some subset of segments $R \subseteq S$ that forms a valid spatial region. We consider a valid spatial region to fit the definition of complex regions as provided by [1]. We term this the *region extraction and verification problem* (REVP).

This paper provides a solution to the REVP that: i) identifies the parts of a scene that form a valid region, ii) identifies the parts of the scene that are invalid according to the definition of complex regions, iii) is efficient in terms of both computation and space requirements, and iv) is suited for implementation in spatial and spatio-temporal databases. Our algorithm takes an arbitrary set of segments and returns two sets of segments: those that form a valid region, and the remaining segments. Furthermore, the segments that form a valid region will be properly annotated for input to existing algorithms for spatial operations such as intersection, topological predicate operations, etc.

## 2 Related Work

Although much work has been completed in the literature with respect to geometric algorithms for spatial data applications, the authors are unaware of any work which directly addresses the problem stated in Section 1. Specifically, no algorithm exists that is applicable to complex regions. Algorithms, such as those presented in [3,4,5,6], detect polygon structures in arrangements of line segments, or create planar maps from arrangements of line segments; however, the REVP is fundamentally different than the problems solved in these papers because (i) there are no restrictions on the polygons detected (i.e., they do not have to be convex, they can contain holes, etc.) (ii) holes and outer cycles of regions must be handled correctly, and (iii) all polygons formed must collectively fit the constraints provided by the definition of complex regions.

We employ the well known plane-sweep algorithmic paradigm in our algorithm. This type of algorithm is original to Shamos and Hoey [7], and a popular version is introduced by Bentley and Ottmann [8], and much additional work has been proposed on the technique [9,10].

## 3 Data Model

*A Halfsegment Representation of Regions:* In this section, we provide an informal type definition. For a formal definition, see [1]. Spatial operation implementations between regions based on the plane sweep algorithm require input to be a region encoded not as a sequence of line segments, but as a sequence of halfsegments. We define the type $halfsegment = \{(s, d, l, r) | s \in segment, d \in bool, l, r \in \mathbb{Z}\}$ where a segment is a straight line segment between two endpoints and $l$ and $r$ are *labels* corresponding to the portion of the embedding space that lies above or to the left of the line that the halfsegment lies on, and the portion that lies below and to the right, respectively. Thus, a halfsegment is said to have two *sides*, a left and right side corresponding to each label. For a halfsegment $h = (s, d, l, r)$, if $d$ is true (false), the smaller (greater) endpoint of $s$ is the *dominating point* of $h$, and $h$ is called a *left* (*right*) *halfsegment*. Hence, each segment $s$ is mapped to two halfsegments $(s, true)$ and $(s, false)$. Given one halfsegment $h$, we denote the halfsegment with identical endpoints and an opposite boolean flag as $h$'s *brother*. We require an order relation on halfsegments. Informally, a complete ordering over halfsegments exists based on their dominating points. If two halfsegments have the same dominating point, then the smaller halfsegment is the one encountered first when rotating a vertical line extending above the dominating point counter-clockwise around the dominating point. A simple polygon is a connected sequence of segments that forms a single cycle. Two simple polygons are *edge-disjoint* if their interiors are disjoint and they possibly share single boundary points but not boundary segments. A face is a simple polygon possibly containing a set of edge-disjoint holes, which are simple polygons, such that these holes do not collectively separate the interior of the face. A complex region is a set of edge-disjoint faces.

*Classifying Invalid Cases:* We make the assumption that halfsegments used as input intersect at endpoints only. This can be enforced with geometric algorithms without affecting the worst case time complexity of our algorithm (Section 4.3).

Based on our data model, we make the observation that all invalid configurations fall into one of two categories: (i) adjacent cycles, when two cycles are not edge-disjoint, and (ii) stick configurations, in which halfsegments do not form a cycle. Figure 2 depicts an example of each case. In Figure 2a, either the right or left cycle may be discarded and a valid region remains. In Figure 2b, Figure 2c, and Figure 2d, the halfsegments forming the sticks must be discarded to form a valid region. These figures respectively illustrate the various forms of the stick configuration that must be handled correctly: (i) disconnected stick configurations (when sticks do not connect to any cycle), (ii) internal stick configurations (when sticks connect to the interior of a cycle), and (iii) external stick configurations (where sticks connect to the exterior of a cycle). Lemma 1 shows that adjacent cycles and stick configurations are the only invalid configurations.

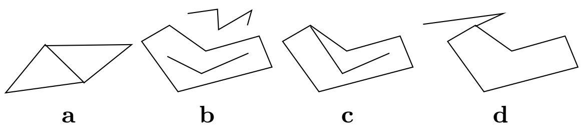


**Fig. 2.** Invalid halfsegment configurations

**Lemma 1.** *The only invalid configurations that arise in the given data model for complex regions are stick configurations and adjacent cycle configurations.*

*Proof Sketch. Consider a region $r$ of straight line segments. Clearly, the removal of any of the segments in $r$ cannot cause segments to intersect at points other than endpoints, but may cause stick or adjacent cycle configurations. Second, consider the addition of new line segments to an already valid region. Because of our definition of regions, and the requirement that line segments only intersect at endpoints, the addition of new segments can only result in the addition of new stick features or new cycles. Therefore, stick and adjacent cycle configurations are the only invalid configurations that must be handled.* □

## 4 Algorithm

Our algorithm removes segments involved in adjacent cycle and stick configurations in an input halfsegment sequence until no such configurations remain. The result is a valid region with properly annotated halfsegments.

Our algorithm operates by removing halfsegments from consideration once it is able to identify them as being part of a valid cycle or stick configuration. Removing halfsegments is achieved by marking their labels to indicate that a halfsegment is part of a valid cycle (in which case one label indicates the interior of a region, and the other indicates the exterior), or that it is part of an invalid configuration (in which case both labels are marked as being invalid). Thus, any adjacent cycles will have some of their halfsegments removed; the result is that adjacent cycles are converted into stick configurations (Figure 3). Therefore, the algorithm is complete when all halfsegments are marked as valid or invalid. The algorithm proceeds in the following steps: (i) find the first halfsegment that is not yet removed from consideration, (ii) determine if the halfsegment lies on the interior of a known face, (iii) discover all other halfsegments that form a cycle or stick configuration containing the current halfsegment and label them, and (iv) repeat until all halfsegments are properly labeled.

### 4.1 Finding and Labeling an Unprocessed Halfsegment

The algorithm begins by finding the least halfsegment $h$ in halfsegment order that is not yet removed from consideration. We say that such a halfsegment is *unprocessed*. We must then determine if $h$ lies on the interior of a known face of a region, or if it lies in the exterior of a region (i.e., $h$ is potentially part of an outer cycle of a face of a region). Therefore, we frame our algorithm in terms of

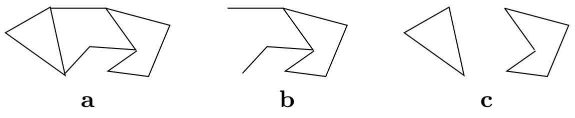


**Fig. 3.** An input to our algorithm containing adjacent cycle configurations (a), the scene after identifying the leftmost cycle as valid (b), and the final valid region (c)

a *plane sweep* algorithm. Recall that a plane sweep algorithm uses an imaginary line that traverses the input halfsegment sequence and each time it encounters a left halfsegment, that halfsegment is placed in the *active list*. The active list is ordered according to the point at which each segment in the active list intersects the sweep line. Each time a right halfsegment is encountered, its corresponding left halfsegment is removed from the active list.

When a new halfsegment $h$ is encountered, we can always determine whether or not it is part of an outer cycle or hole cycle by looking at the labels of halfsegment $b$ that will be immediately below $h$ in the active list. This information allows us to move on to the next step (the *cycle walk* in Section 4.2) and identify all halfsegments that form a cycle with $h$. $b$ will always have been previously processed due to the halfsegment ordering.

Once a cycle walk has been performed, the plane sweep portion of the algorithm resumes and traverses the halfsegment sequence until a partially processed (Section 4.2) or unprocessed halfsegment is found. If a halfsegment is found that is unprocessed, then the cycle walk portion of the algorithm is executed. If the halfsegment is partially processed, then it is marked as being part of an invalid configuration, and the plane sweep portion of the algorithm continues. The reasoning is given in the following lemma:

**Lemma 2.** *A partially processed halfsegment encountered in the plane sweep portion of the algorithm is always part of a stick configuration.*

*Proof Sketch. According to Lemma 4, the cycle walk portion of the algorithm will convert any adjacent cycle configurations to stick configurations. Furthermore, the only time a halfsegment is partially processed is during the cycle walk portion of the algorithm. Therefore, since a partially processed halfsegment has already been visited by a cycle walk, it must be part of a stick configuration since the cycle walk will correctly identify halfsegments that belong to a valid cycle.* □

## 4.2 Walking the Cycle

Once an unprocessed halfsegment $h$ is found and it is known whether $h$ lies in the interior of a face of a region or not, the next step is to identify all halfsegments that form the cycle or stick configuration that $h$ is part of. In order to identify the segments that form a cycle containing $h$, we make the observation that according to the definition of complex regions, each cycle that forms part of the boundary of a region separates the embedding space into three, disjoint point sets: the interior, exterior, and boundary. Therefore, given a halfsegment $h$ and the knowledge of upon which side of the halfsegment the interior of the cycle lies, the adjacent halfsegment in the cycle can be found by rotating $h$ around its submissive point through the interior of the cycle until a new halfsegment $h_{next}$ is found. Because $h$ was rotated through the interior of a cycle, the next halfsegment encountered must also bound the interior of that cycle and we can trivially determine on which side of the halfsegment the interior lies for labeling. This is repeated until a stopping condition (see below) occurs. Due to halfsegment ordering, a clockwise rotation around submissive points is always used.

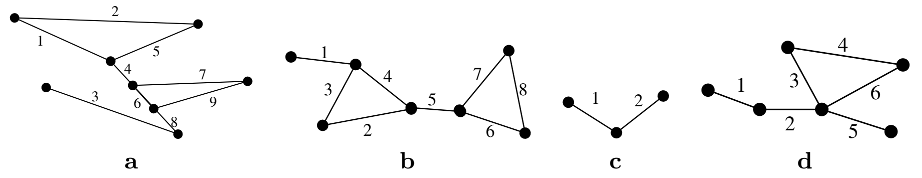


**Fig. 4.** Instances of stick configurations

We denote the procedure of visiting halfsegments in a cycle in cyclic order and assigning their appropriate labels as a *cycle walk*.

Let the notation $1_l, 1_r$ to refer to the left and right halfsegment of segment 1, etc. A cycle walk beginning at halfsegment $h$ proceeds until one of three stopping conditions occurs: (i) $h$ is encountered a second time, (ii) the brother of $h$, $h_b$ is encountered, or (iii) a halfsegment $j$ is encountered that is already correctly labeled. In the first case, encountering $h$ a second time during a cycle walk indicates that the complete and valid cycle has been walked (e.g., walking of the leftmost cycle in Figure 4a by visiting $1_l$ then $5_l$ then $2_r$). The second case deals with a *leading stick configuration*. In this case, the exterior of a cycle (or cycles) is walked (e.g., a cycle walk on Figure 4b will visit $1_l$, then $4_l$ then continue until it reaches $1_r$). In this case, any segment that has had both corresponding halfsegments visited is a stick. Any other halfsegments cannot be marked as valid or invalid, so they are labeled as being *partially processed* (we know upon which side of the halfsegment the exterior lies, but cannot ensure that the segments are part of a valid cycle). The final case indicates that a stick configuration was formed due to the previous identification of a valid cycle. We denote this case a *trailing stick configuration*, and it arises in Figure 4a. Internal stick configurations are identified when the first stopping condition occurs, and both corresponding halfsegments forming internal stick configurations are visited.

**Lemma 3.** *The cycle walk portion of the algorithm correctly identifies segments belonging to stick configurations and segments that form valid cycles.*

*Proof Sketch. A halfsegment rotation through a cycle's interior guarantees that halfsegments that bound the cycle will be visited. It is clear that segments forming internal stick configurations will have both corresponding halfsegments visited, and will therefore be marked as invalid. In order to classify the boundary halfsegments correctly, we must be able to identify if a cycle walk or an external walk occurred. In a cycle walk, the halfsegment h that began the cycle walk will be encountered before its brother $h_b$. Otherwise, an external walk must have been performed and h is the first halfsegment in a leading stick configuration. This follows directly from the halfsegment ordering and a clockwise rotation around endpoints. Therefore, we can discern the cycle walk scenarios.* □

**Lemma 4.** *The cycle walk portion of the algorithm must only be able to identify stick configurations and valid cycles, not adjacent cycle configurations.*

*Proof Sketch. Once a halfsegment has been identified as being part of a valid cycle or is identified as being invalid, then it is removed from consideration from the* *algorithm. In the case of adjacent cycles, a cycle walk will identify one cycle, and remove its halfsegments from consideration. This effectively converts adjacent cycles to a valid cycle and a stick configuration (Figure 3). Therefore, the cycle walk algorithm must only be able to identify valid cycles and stick configurations.* □

## 4.3 Discussion

A plane sweep algorithm for finding line segment intersections can be implemented in $O(n \lg n + k)$ time complexity and $O(n + k)$ space complexity for $n$ segments and $k$ segment intersections. We begin with the assumption that line segments do not intersect; therefore, we have $O(n \lg n)$ and $O(n)$ time and space complexity, respectively. Each halfsegment is visited at most twice in the algorithm, once to partially process it, and once to fully process it. Whenever a halfsegment is marked as fully processed, we mark its brother identically. Thus, a logarithmic search technique is used to locate the brother. The cycle walk portion of the algorithm requires us to find a halfsegment in cyclic order from a given halfsegment. This can be done in logarithmic time using a binary search technique that takes advantage of halfsegment ordering. Therefore, the

**Algorithm 1:** A pseudo-code implementation of the proposed algorithm.

```
   Input: Sequence of non-labeled halfsegments $H$
   Output: Sequence of labeled halfsegments $J$
1  while not end of plane sweep do
2      Advance sweep line to $h$, the left-most unprocessed or partially processed halfsegment.;
3      Find halfsegment $j$ below $h$ in the sweep line active list;
4      var $isOutercycle \leftarrow True$;
5      var $interiorIsAboveHalfsegment \leftarrow True$;
6      var $currCycle \leftarrow 5$;  var $unvisitedLabel \leftarrow 0$;
7      var $exteriorLabel \leftarrow 3$;  var $interiorLabel \leftarrow 4$;
8      var $invalidAndInExterior \leftarrow 1$;  var $invalidAndInInterior \leftarrow 2$;
9      var $invalidLabel \leftarrow invalidAndInExterior$;
       // check to see if $h$ lies in the interior of the face bordered by $j$
10     if $j.labelAbove \neq exteriorLabel$ OR $j.labelAbove = invalidAndInInterior$ then
11         $isOutercycle \leftarrow False$;
12         $invalidLabel \leftarrow invalidAndInInterior$;
       // handle the case that $h$ is partially processed. If $h$ has an exterior on one
          side, and an unvisited label on the other, it is partially processed.
13     if ($h.labelAbove = unvisitedLabel$ AND $h.labelBelow = exteriorLabel$) OR
       ($h.labelAbove = exteriorLabel$ AND $h.labelBelow = unvisitedLabel$) then
           // Fully process $h$, setting the labels of both sides to the invalid label
14         $h.aboveLabel \leftarrow h.belowLabel \leftarrow invalidLabel$;
15         Find $h_b$, the brother of $h$;
16         $h_b.aboveLabel \leftarrow h_b.belowLabel \leftarrow invalidLabel$;
17     else
           // Walk the cycle
18         if $isOutercycle$ then
19             $h.labelAbove \leftarrow currCycle$;  $h.labelBelow \leftarrow exteriorLabel$;
20         else $h.labelAbove \leftarrow exteriorLabel$;  $h.labelBelow \leftarrow currCycle$;
21         $interiorIsAboveHalfsegment \leftarrow isOuterCycle$;
22         $prev \leftarrow h$;
23         $k \leftarrow findNextInCycle(h)$;
24         $visitedStack \leftarrow emptystackofhalfsegments$;
           // (continued)
70
71
```

**Algorithm 1**: *(continued).*

```
25 while $True$ do
       // find if the interior of the face is above this halfsegment
26     if $\neg sameInteriorAbove(prev, k)$ then
27         $interiorIsAboveHalfsegment \leftarrow \neg interiorIsAboveHalfsegment$;
       // Check to see if the cycle walk is complete
28     if $k = h$ then
           // Convert the partially processed labels to fully processed labels
29         foreach $halfsegmenti\ in\ visitedStack$ do
30             if $i.labelAbove > interiorLabel$ then
31                 $i.labelAbove \leftarrow interiorLabel$;
32             else if $i.labelAbove = unvisitedLabel$ then
33                 $i.labelAbove \leftarrow exteriorLabel$;
34             if $i.labelBelow > interiorLabel$ then
35                 $i.labelBelow \leftarrow interiorLabel$;
36             else if $i.labelBelow = unvisitedLabel$ then
37                 $i.labelBelow \leftarrow exteriorLabel$;
38             Find $i_b$, the brother of $i$;
39             $i_b.aboveLabel \leftarrow i.aboveLabel$;
40             $i_b.belowLabel \leftarrow i.belowLabel$;
41         goto ENDOFCYCLEWALK;
       // Now check for the invalid cases
       // First invalid case: we encounter the same segment twice in a cycle walk
42     Find $k_b$, the brother of $k$;
43     if $k_b.labelAbove = currCycle\ OR\ k_b.labelBelow = currCycle\ AND\ k \neq h_b$ then
           // Mark the halfsegment as invalid
44         $k.labelAbove \leftarrow k.labelBelow \leftarrow invalidLabel$;
45         $k_b.labelAbove \leftarrow k_b.labelBelow \leftarrow invalidLabel$;
46     // Second invalid case: we started on a stick and performed an exterior walk
       // Third invalid case: we encounter a fully processed halfsegment, which means we
          performed an exterior cycle walk on some halfsegments
47     else if $(k = h_b)\ OR\ (k.labelAbove \neq currCycle\ AND\ k.labelAbove \neq unvisitedLabel$
       $AND\ k.labelBelow \neq currCycle\ AND\ k.labelBelow \neq unvisitedLabel)$ then
           // Flip the labels since we have walked the exterior
48         foreach $halfsegment\ i\ in\ visitedStack$ do
49             $tmp \leftarrow i.labelAbove$;
50             $i.labelAbove \leftarrow i.labelBelow$;
51             $i.labelBelow \leftarrow tmp$;
52             if $i.labelAbove = i.labelBelow$ then
                   // Visited the halfsegment twice in a cycle walk. It is a stick
53                 $i.labelAbove \leftarrow i.labelBelow \leftarrow invalidLabel$;
54             else
                   // Assign the label that is not on the exterior side to unknown label
55                 if $i.labelAbove > interiorLabel$ then $i.labelAbove \leftarrow unvisitedLabel$;
56                 if $i.labelBelow > interiorLabel$ then $i.labelBelow \leftarrow unvisitedLabel$;
           // Mark $h$ and $h_b$as being invalid
57         $h.labelAbove \leftarrow h.labelBelow \leftarrow invalidLabel$;
58         $h_b.labelAbove \leftarrow h_b.labelBelow \leftarrow invalidLabel$;
59         goto ENDOFCYCLEWALK;
60     else
           // If we get here, then there is nothing wrong. process this halfsegment.
61         if $interiorIsAboveHalfsegment$ then
62             $k.labelAbove \leftarrow currCycle$;
63             if $k.labelBelow = unvisitedLabel$ then $k.labelBelow \leftarrow exteriorLabel$;
64         else
65             $k.labelBelow \leftarrow currCycle$;
66             if $k.labelAbove = unvisitedLabel$ then $k.labelAbove \leftarrow exteriorLabel$;
       // Set up for the next iteration of the cycle walk loop
67     $prev \leftarrow k$;
68     $k \leftarrow findNextInCycle(prev)$;
69 ENDOFCYCLEWALK;
```

complete algorithm is bounded by $O(n \lg n)$ in time complexity and $O(n)$ space complexity for an input configuration of $n$ segments. Running a line segment intersection algorithm to ensure that segments intersect only at endpoints takes $O(n \lg n + k)$ time complexity, and in practice does not affect running time in typical cases.

Algorithm 1 shows the final algorithm as it has been described. The final step is to ensure correctness. Our definition of correctness is that our algorithm can effectively handle all invalid spatial configurations, and that it returns a valid region. Lemmas 1-4 describe the possible invalid cases that must be handled by the algorithm, and show that the algorithm handles the cases correctly. Therefore, our algorithm will always return a valid region given a valid input:

**Theorem 1.** *Given valid input, the proposed algorithm will identify, correctly label, and return halfsegments forming a valid region.*

*Proof Sketch. Lemmas 1-4 indicate the cases that the algorithm must handle and shows how the algorithm correctly handles each case.* □

## 5 Conclusion

In this paper, we have identified the REVP as an important problem in the growing fields of spatio-temporal and moving objects databases, and provided examples of the problem in traditional spatial applications such as remote sensing and geo-sensor networks. We have developed, implemented, and presented an efficient $O(n \lg n)$ time complexity algorithm that can be used to solve this problem that can be incorporated into spatial systems. Furthermore, our algorithm uses an input format that is common in spatial algorithms and spatial data representations so that it can be easily incorporated into existing systems.

## References

1. Schneider, M., Behr, T.: Topological Relationships between Complex Spatial Objects. ACM Trans. on Database Systems (TODS) 31(1), 39–81 (2006)
2. Forlizzi, L., Güting, R.H., Nardelli, E., Schneider, M.: A Data Model and Data Structures for Moving Objects Databases. In: ACM SIGMOD Intl. Conf. on Management of Data, pp. 319–330 (2000)
3. Amato, N.M., Goodrich, M.T., Ramos, E.: Computing Faces in Segment and Simplex Arrangements. In: ACM Symp. on Theory of Computing, pp. 672–682 (1995)
4. Asano, T., Guibas, L., Tokuyama, T.: Walking on an Arrangement Topologically. In: ACM Annual Symp. on Computational Geometry, pp. 297–306 (1991)
5. Edelsbrunner, H., Guibas, L.J., Sharir, M.: The Complexity of Many Faces in Arrangements of Lines of Segments. In: ACM Annual Symp. on Computational Geometry, pp. 44–55. ACM Press, New York (1988)

6. Ferreira, A., Fonseca, M.J., Jorge, J.A.: Polygon Detection from a Set of Lines. In: Encontro Portugues de Computacao Grafica, pp. 159–162 (2003)
7. Shamos, M., Hoey, D.: Geometric Intersection Problems. In: IEEE Symp. on Foundations of Computer Science (1976)
8. Bentley, J., Ottmann, T.: Algorithms for Reporting and Counting Geometric Intersections. IEEE Trans. on Computers C 28, 643–647 (1979)
9. Nievergelt, J., Preparata, F.P.: Plane-Sweep Algorithms for Intersecting Geometric Figures. Communications of the ACM (CACM) 25, 739–747 (1982)
10. Balaban, I.J.: An Optimal Algorithm for Finding Segments Intersections. In: ACM Annual Symp. on Computational Geometry, pp. 211–219. ACM Press, New York (1995)

# Identifying the Most Endangered Objects from Spatial Datasets

Hua Lu and Man Lung Yiu

Department of Computer Science, Aalborg University, Denmark
{luhua,mly}@cs.aau.dk

**Abstract.** Real-life spatial objects are usually described by their geographic locations (e.g., longitude and latitude), and multiple quality attributes. Conventionally, spatial data are queried by two orthogonal aspects: spatial queries involve geographic locations only; skyline queries are used to retrieve those objects that are not dominated by others on all quality attributes. Specifically, an object $p_i$ is said to dominate another object $p_j$ if $p_i$ is no worse than $p_j$ on all quality attributes and better than $p_j$ on at least one quality attribute. In this paper, we study a novel query that combines both aspects meaningfully. Given two spatial datasets $P$ and $S$, and a neighborhood distance $\delta$, the *most endangered object query* (MEO) returns the object $s \in S$ such that within the distance $\delta$ from $s$, the number of objects in $P$ that dominate $s$ is maximized. MEO queries appropriately capture the needs that neither spatial queries nor skyline queries alone have addressed. They have various practical applications such as business planning, online war games, and wild animal protection. Nevertheless, the processing of MEO queries is challenging and it cannot be efficiently evaluated by existing solutions. Motivated by this, we propose several algorithms for processing MEO queries, which can be applied in different scenarios where different indexes are available on spatial datasets. Extensive experimental results on both synthetic and real datasets show that our proposed advanced spatial join solution achieves the best performance and it is scalable to large datasets.

## 1 Introduction

Real-life spatial objects (e.g., hotels) are not only associated with geographic locations but also with multiple quality attributes (e.g., price and star). Conventionally, spatial objects are retrieved by using two orthogonal query types. Spatial queries (e.g., nearest neighbor query, closest pair query) select objects solely based on their geographic locations, or derived measures such as distances. These queries fail to utilize the rich information captured by quality attributes.

On the other hand, the skyline query [1] is a powerful multi-criteria optimization tool for retrieving objects based on their quality attributes. Specifically, this query returns the objects that are not dominated by others on all quality attributes. An object $p_i$ is said to dominate another object $p_j$ if $p_i$ is no worse than $p_j$ on all quality attributes and better than $p_j$ on at least one quality attribute.

Unfortunately, skyline queries focus only on the quality attributes, disregarding the importance of spatial distances among the objects.

**Motivation of Retrieving the Most Endangered Objects**
Nowadays, spatial decision support systems need to combine both the location and quality aspects of spatial objects in a meaningful way to retrieve desired objects for the users. This is especially true for a practical application that identifies objects in endangered positions and conducts appropriate planning for them. As an example, suppose that a hotel chain is facing financial challenges and it plans to shut down one of its hotels. Intuitively, a hotel is unlikely to make profit if it is located close to a large number of competitor hotels that have dominating advantages on all quality attributes. Such a hotel may be considered to be shut down. Note that the business of a hotel is not significantly affected by competitor hotels that are far away. Thus, two hotels are considered as geographically close if their distance is within a given neighborhood distance $\delta$.

| hotel | price | star |
|---|---|---|
| $s_1$ | \$200 | 4 |
| $s_2$ | \$100 | 2 |
| $s_3$ | \$250 | 5 |
| $s_4$ | \$160 | 3 |
| $s_5$ | \$160 | 3 |

**Fig. 1.** Candidate set $S$

| hotel | price | star |
|---|---|---|
| $h_1$ | \$180 | 4 |
| $h_2$ | \$200 | 3 |
| $h_3$ | \$200 | 5 |
| $h_4$ | \$250 | 3 |
| $h_5$ | \$200 | 5 |
| $h_6$ | \$220 | 4 |
| $h_7$ | \$100 | 2 |
| $h_8$ | \$150 | 3 |
| $h_9$ | \$200 | 5 |
| $h_{10}$ | \$160 | 4 |

**Fig. 2.** Competitor set $P$

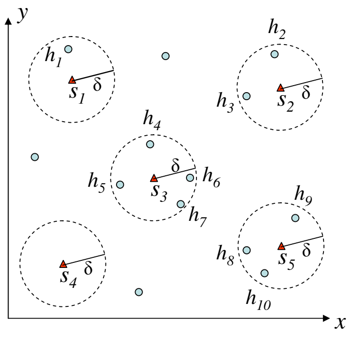


**Fig. 3.** Locations of hotels

Specifically, given a spatial dataset $P$ (of competitors), a spatial dataset $S$ (of candidates), and a neighborhood distance $\delta$, the *most endangered object query* (MEO) query returns the object $s \in S$ such that within the distance $\delta$ from $s$, the number of objects of $P$ that dominate $s$ is maximized.

Figure 1 lists the quality attributes (price and star) of hotels in a candidate set $S$ that belongs to a hotel chain. Similarly, Figure 2 shows the quality attributes of the set $P$ of competitor hotels. Here, lower prices and higher stars are preferred. For example, the candidate $s_1$ is dominated by the competitors $h_1, h_3, h_5$. The locations of all hotels are shown in Figure 3, where competitors are drawn as dots, candidates are shown as triangles, and the neighborhood distance $\delta$ is indicated by dashed circles. Within the distance $\delta$, the hotel $s_1$ is only dominated by $h_1$. $s_2$ is dominated by no hotels, even though it is surrounded by $h_2$ and $h_3$. Next, $s_3$ is dominated by the nearby $h_5$ only; $s_5$ is dominated by both $h_8$ and $h_{10}$. Therefore, the MEO query returns the hotel $s_5$ as the result since it is dominated by the largest number of competitors within the spatial distance $\delta$.

In addition to the business planning application described above, MEO queries also provide useful results for other fields. For online war games (e.g., War of

Warcraft), $P$ is the set of enemy troop locations and $S$ is the set of ally troop locations. Each troop can be described by multiple quality attributes like solider number, equipment level, etc. The MEO query can then be used to identify the most endangered troop of $S$ that need additional combat support. In wild animal protection, $S$ denotes the set of endangered species and $P$ represents their enemies. Quality attributes refer to abilities such as strength, agility, and stamina. The MEO query helps to identify the animal most worthy of extra protection.

**Deficiency of Existing Solutions**
It is noteworthy that neither a conventional spatial query nor a skyline query alone will return the same result as a MEO query. A skyline query on the competitor set followed by a spatial query with respect to all candidates does not help either, because some candidates are dominated by non-skyline competitors. Referring to our earlier example, the candidate $s_1$ is dominated by the competitor $h_1$ which however is not a skyline object among the competitors (since $h_1$ is dominated by $h_{10}$).

In Section 2.3, we will elaborate two straightforward solutions that correctly process the MEO query: (i) an RDBMS solution, and (ii) a multi-step R-tree based solution. Those solutions do not fully exploit the characteristics of the MEO query so they incur high processing cost.

**Our Contributions**
Motivated by these observations, we propose several algorithms for processing MEO queries. These algorithms can be applied in different scenarios where different indexes are available on input datasets. Our first algorithm is a baseline iterative approach which needs only an R-tree index on the dataset $P$. Our next two algorithms require that the dataset $S$ is indexed by an R-tree $R_S$, and the dataset $P$ is indexed by an aggregate R-tree $R_P$. In the aggregate R-tree [12], each node entry stores a count of objects in its subtree. Then the query is processed by a depth-first or best-first search on the tree $R_S$, based on efficient upper bound counting techniques using $R_P$. Our last algorithm requires the same indexes as the previous two, but it evaluates the query in a spatial join manner.

The remainder of this paper is organized as follows. Section 2 formally defines the MEO query and briefly reviews the related work. Section 3 develops our algorithms for processing MEO queries. Section 4 presents extensive experimental results of our proposals on both synthetic and real data. Finally, Section 5 concludes the paper and discusses future research directions.

## 2 Preliminaries

### 2.1 Problem Statement

We assume that all quality attributes are numeric and each attribute domain is totally ordered. Let $c$ be the number of quality attributes. A *quality vector* is a point $\psi$ in the $c$-dimensional space $\mathbb{R}^c$, where each dimension refers to a quality attribute. $\psi[i]$ denotes the $i$-th (quality) attribute value of $\psi$.

Without the loss of generality, we assume that smaller values are preferred to larger ones in quality attributes throughout this paper. According to [1], a quality vector $\psi$ *dominates* another one $\psi'$ (denoted as $\psi \prec \psi'$) when:

$$(\exists\, 1 \leq i \leq c,\ \psi[i] < \psi'[i]) \wedge (\forall\, 1 \leq i \leq c,\ \psi[i] \leq \psi'[i]) \tag{1}$$

A *location* is a pair $(x, y)$ in the Euclidean space $\mathbb{R}^2$, where $x$ and $y$ are coordinate values. A *spatial object* $o = \langle loc, \psi \rangle$ consists of both a location $o.loc$ and a quality vector $o.\psi$. The notation $dist(o, o')$ denotes the Euclidean distance between the locations of the spatial objects $o$ and $o'$. Given two spatial objects $o$ and $o'$, $o$ is said to *dominate* $o'$ when $o.\psi \prec o'.\psi$.

**Definition 1 (Neighborhood Dominating Score).** *Given a spatial object set $P$, a spatial object $s$, and a neighborhood distance $\delta$, the neighborhood dominating score of $s$ on $P$ with respect to $\delta$ is defined as:*

$$\Phi_{P,\delta}(s) = |\{o \in P \mid dist(o, s) \leq \delta, o.\psi \prec s.\psi\}|$$

Whenever the context becomes clear, we drop the subscripts of $\Phi(s)$. We then define the *most endangered object query* (MEO) as follows. Our objective is to design an I/O-efficient solution for processing MEO queries on large datasets.

**Definition 2 (Most Endangered Object Query).** *Given two spatial object sets $P$ and $S$, for competitors and candidates respectively, and a neighborhood distance $\delta$, the* **most endangered object query** *(MEO) returns from $S$ an object $s$ such that $\Phi_{P,\delta}(s)$ is maximized, i.e., $\forall\, s' \in S$, $\Phi_{P,\delta}(s) \geq \Phi_{P,\delta}(s')$*

## 2.2 Related Work

**Spatial Join**

Given a distance bound $\delta$, and two spatial datasets $S$ and $P$, *the $\delta$-distance join* retrieves each pair $\langle s, p \rangle$ (where $s \in S$ and $p \in P$) such that their Euclidean distance $dist(s, p)$ is within $\delta$. The *R-tree join* (RJ) [2] can be applied to evaluate the $\delta$-distance join if both $S$ and $P$ are indexed by R-trees $R_S$ and $R_P$ respectively. RJ first examines the entries in the root nodes of $R_S$ and $R_P$. If an entry $e_S$ (of the tree $R_S$) and an entry $e_P$ (of the tree $R_P$) satisfies $mindist(e_S, e_P) \leq \delta$, then the subtrees of $e_S$ and $e_P$ may contain some objects within $\delta$. In that case, RJ is recursively applied on the subtrees of $e_S$ and $e_P$. Eventually, RJ reaches the leaf level and reports the pairs of objects that are within $\delta$. Efficient $\delta$-distance join algorithms on high-dimensional data have been studied in [8]. Zhu et al. [19] proposed the *top-k spatial join* for computing $k$ objects of $S$ that intersect with the largest number of objects in $P$.

The above studies consider only the spatial relationship between the objects in $S$ and $P$, but not their dominance relationship on quality attributes of objects in our MEO query.

**Location Selection Queries**

In the literature, various constraints have been combined with conventional spatial queries in order to select semantically optimal locations or objects.

Du et al. [5] proposed the optimal-location query. Given a site set $S$, a weighted object set $O$, and a spatial region $Q$, the optimal-location query returns a location in $Q$ with the maximum influence. The influence of a location $l$ is defined as the total weights of objects in $O$, each of which has $l$ as its nearest neighbor in the set $S \cup \{l\}$. Using the same influence definition, Xia et al. [15] formulated a different top-$k$ most influential spatial sites query, which returns $k$ sites (from $S$ and within $Q$) having the highest influences. In the same context, Zhang et al. [17] proposed the min-dist optimal-location query. Rather than maximizing influence, this query selects from $Q$ a location $l$ which minimizes the average distance from every object in $O$ to its nearest site in $S \cup \{l\}$. All these optimal-location queries differ from our MEO query in the sense that they do not consider multi-dimensional dominance relationship among the non-spatial quality attributes of the objects. This renders their solutions inapplicable to our problem.

Yiu et al. [16] formalized the top-$k$ spatial preference query, which returns the $k$ spatial objects with the highest ranking scores, based on the feature qualities in their spatial proximity. Such score functions, however, do not support multi-dimensional dominance relationship as in our MEO query.

Li et al. [9] proposed Dominant Relationship Analysis (DRA), for discovering the dominant relationship between products and potential consumers. To efficiently answer different analysis queries of DRA, the authors proposed a datacube structure, named *DADA*, which stores the dominant relationships in the way supporting ordered access and compressing. Li et al. [10] combined dominance relationship with spatial distance and defined complex location selection problems. However, its solution cannot be applied to solve our problem. Only one dataset is considered in [10], from which desirable objects are selected. In contrast, our problem involves two datasets with different practical semantics, and aims at selecting an object (from the candidate dataset $S$) with the highest score defined with respect to the competitor dataset $P$.

**Skyline Queries in Spatial and Spatiotemporal Settings**
Skyline queries has been adopted in spatial and spatiotemporal database to define specific problems. Huang and Jensen [6] proposed an in-route skyline query for location-based services. When moving along a pre-defined road route towards her/his destination, a user may visit points of interest in the network. Points to visit are selected in terms of multiple distance-related preferences like detour and total traveling distance. The authors optimize such selections using skyline queries involving specific interesting dimensions.

Sharifzadeh and Shahabi [14] studied the spatial skyline query, which is in fact a specialized version of the dynamic skyline query [13]. Given a set of query points $Q = \{q_1, \ldots, q_n\}$ and two points $p$ and $p'$, $p$ is said to spatially dominate $p'$ iff $dist(p, q_i) \leq dist(p', q_i)$ for any $q_i \in Q$ and $dist(p, q_i) < dist(p', q_i)$ for at least one $q_i \in Q$. The spatial skyline of a set of points $P$ is the subset of all points not spatially dominated by any other point of $P$. Observe that such queries consider only spatial attributes but not any non-spatial quality attributes.

Huang et al. [7] defined continuous skyline query in a spatiotemporal context. A spatial object $p$ dominates another object $p'$ with respect to a query location $q$, if $p$ is closer to $q$ than $p'$ and $p$ dominates $p'$ on all non-spatial attributes. A continuous skyline query then maintains all spatial objects not dominated by any others, while the query $q$ is continuously moving along a specified trajectory in the Euclidean space. Using a similar setting, Zheng et al. [18] addressed how to compute the valid scope for such a query result without knowing the movement pattern of the object.

Our MEO query is also different from the constrained skyline query [13], in which the objects being considered is restricted by a given constraint region in the domain space of (multiple) quality attributes. In contrast, the spatial distance constraint $\delta$ employed in the MEO query is only used in the spatial domain but not on quality attributes.

The main difference of the MEO query from the above studies is that the MEO query is not a skyline problem. Recall from the motivation example in the Introduction that a non-skyline object (e.g., hotel $h_1$) in the competitor set can still dominate a candidate object (e.g., hotel $s_1$) within its spatial neighborhood.

## 2.3 Straightforward Solutions

In this section, we describe two straightforward solutions for evaluating the MEO query, and then discuss their drawbacks.

**RDBMS Solution (SQL)**

In fact, the MEO query can be expressed by the following SQL statement (see Figure 4), and thus it can be executed in any existing commercial RDBMS. Here, we assume that the input datasets are stored in two relational tables `S` and `P`, which share the same schema. The attribute `id` is the identifier of a tuple. The attributes `x` and `y` represent spatial coordinates; whereas the attributes `psi_1`, `psi_2`, etc, are the quality attributes. The query parameter $\delta$ is translated to the value `delta` in the SQL query.

In the following query, we first join the tuples of the tables `S` and `P`. Definition 1 is expressed by the join condition in the WHERE clause: the first line refers to the neighborhood distance constraint, and the last two lines represent the dominance comparison. After that, the intermediate join result set is partitioned into groups based on its `id` in `S`. Then, the count of each group is computed and the `id` of the largest group (together with its count) is returned as the result.

```
SELECT      S.id, COUNT(*)
FROM        S, P
WHERE       (S.x-P.x)*(S.x-P.x)+(S.y-P.y)*(S.y-P.y)<=delta*delta
            AND ( P.psi_1<=S.psi_1 AND P.psi_2<=S.psi_2 AND ... )
            AND ( P.psi_1< S.psi_1 OR  P.psi_2< S.psi_2 OR  ... )
GROUP BY    S.id
ORDER BY    COUNT(*)       DESC LIMIT 1
```

**Fig. 4.** Expression of MEO Query in SQL

The main disadvantage of this method is that it incurs very high execution time in existing RDBMS. Even though typical indexes (e.g., $B^+$-trees and hash indexes) may be used by the RDBMS engine, it cannot fully exploit the complex join condition (shown in the WHERE clause) for optimizing the search cost.

**Multi-step R-tree based Solution (3Step)**
Another straightforward method for the MEO query is as follows, assuming that the datasets $S$ and $P$ are indexed by two R-trees $R_S$ and $R_P$ respectively.

In the first step, this method performs the $\delta$-distance join by applying the RJ algorithm [2] on those two trees, in order to obtain the pairs $\langle s, p\rangle$ that are within the $\delta$ distance. In the second step, any pair $\langle s, p\rangle$ is pruned if it does not satisfy $p.\psi \prec s.\psi$. In the third step, the remaining pairs are assigned into groups according to $s.id$, and the object $s$ having the largest group count is reported as the result.

The drawback of this method is that the first step incurs a high cost at a large $\delta$ value, regardless of the pruning effectiveness of the second step.

## 3 Algorithms for Most Endangered Object Queries

We in this section detail our algorithms for processing most endangered object queries. We first present the baseline approach that carries out an iterative search on all candidate objects without any index on them. Then improved algorithms are presented with specific index requirements. Table 1 lists the notations to be used throughout the paper.

**Table 1.** Table of Notations

| Notation | Meaning |
|---|---|
| $P$ | the set of objects for competitors |
| $S$ | the set of objects for candidates |
| $\psi \prec \psi'$ | a quality vector $\psi$ dominates another one $\psi'$ |
| $dist(o, o')$ | Euclidean distance between two objects $o$ and $o'$ |
| $mindist(e, e')$ | minimum distance between two R-tree entries $e$ and $e'$ |
| $\Phi(s)$ | neighborhood dominating score of an object $s$ |
| $\odot(s, \delta)$ | a circular region with center $s$ and radius $\delta$ |
| $\Xi(e, \delta)$ | $\delta$-Minkowski region of an R-tree entry e |

### 3.1 A Baseline Approach: Iterative Search Algorithm

In this section, we assume that the dataset $P$ is indexed by an R-tree $R_P$ and the dataset $S$ is not indexed. We first present a basic algorithm for computing the score $\Phi(s)$ of an object $s \in S$, and then apply it iteratively on each object in order to obtain the final result.

**ObjectScore** (see Algorithm 1) is a recursive algorithm for computing the $\Phi(s)$ value of the object $s$ with respect to the objects in the subtree of the entry $e_P$ (of the R-tree $R_P$). The input parameter $\delta$ represents the distance threshold.

At line 1, the counter $v$ is used to maintain the value of $\Phi(s)$. In case $e_P$ is a leaf entry (line 2), we check whether its distance to $s$ is within $\delta$ and its quality vector dominates that of $s$. If so, then the counter $v$ is incremented. When $e_P$ is a non-leaf entry (line 5), we read its the child node, and recursively process each of its entry $e'_P$ if the minimum distance $mindist(e'_P, s)$ from $e'_P$ to $s$ is within $\delta$.

**Algorithm 1. ObjectScore**(Object $s$, Entry $e_P$ of the R-tree $R_P$, Distance $\delta$)

```
1: $v := 0$
2: if $e_P$ is a leaf entry then
3:     if $dist(e_P, s) \leq \delta$ and $e_P.\psi \prec s.\psi$ then
4:         $v := 1$
5: else                                              ▷ $e_P$ is a non-leaf entry
6:     read the child node $CN$ pointed to by $e_P$;
7:     for each entry $e'_P$ in $CN$ do
8:         if $mindist(e'_P, s) \leq \delta$ then
9:             $v := v+$ObjectScore$(s, e'_P, \delta)$
10: return $v$
```

We then propose the iterative search (**IS**) for processing the MEO query. Its pseudo code is shown in Algorithm 2. It takes as input (i) an R-tree on competitor set $P$, (ii) the candidate object set $S$, and (iii) the distance $\delta$. The object $meo$ is used to keep track the result object found so far, and the value $\gamma$ corresponds to the score of $meo$. At line 1, we initialize $meo$ to null and $\gamma$ to 0 respectively. For each location $s$ of the set $S$, the algorithm applies the **ObjectScore** function on the root of R-tree $R_P$ (lines 2–3) to obtain the score $\Phi(s)$ of $s$. If $\Phi(s)$ is higher than $\gamma$, then the result and its score will be updated (lines 4–6). Finally, the algorithm returns the object $meo$ as the result.

The IS algorithm is able to exploit the main-memory buffer better if it processes all the locations of $S$ via a locality-preserving order. Thus, we develop the algorithm IS-Hil, which first applies external sorting on the locations in $S$ based on the Hilbert ordering [4,11], and then processes them by IS.

**Algorithm 2. IS**(R-tree $R_P$ on $P$, Object set $S$, Distance $\delta$)

```
1: $meo$ :=null; $\gamma := 0$
2: for each object $s \in S$ do
3:     $\Phi(s)$ :=ObjectScore$(s, R_P.root, \delta)$
4:     if $\Phi(s) > \gamma$ then
5:         $\gamma := \Phi(s)$
6:         $meo := s$
7: return $meo$
```

## 3.2 Aggregate R-Tree Search Algorithms

Observe that IS algorithm processes every object once in the set $S$. We now propose to index the set $S$ by an R-tree $R_S$ and develop an efficient method to prune unqualified subtrees of $R_S$ that cannot contribute to the result.

In order to support efficient counting operations, we index the dataset $P$ by an aggregate COUNT R-tree $R_P$ [12]. Specifically, each non-leaf entry $e_P$ of the tree $R_P$ stores an additional *count* value, denoted as $e_P.count$, which is equal to the number of objects in the subtree of $e_P$.

**Derivation of Upper Bound Score**

Suppose that $e_S$ is a non-leaf entry of the tree $R_S$. Figure 5 shows the spatial extent of $e_S$ as a rectangular region. We wish to derive an upper bound score $\Phi^+(e_S)$ of $e_S$ such that $\Phi(s) \leq \Phi^+(e_S)$ for any object $s$ in the subtree of $e_S$.

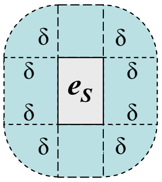


**Fig. 5.** $\Xi(e_S, \delta)$

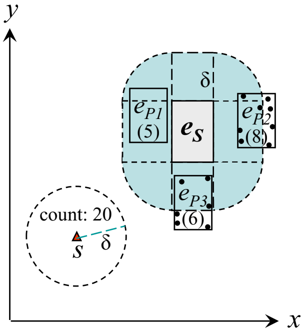


**Fig. 6.** Pruning Rule 1

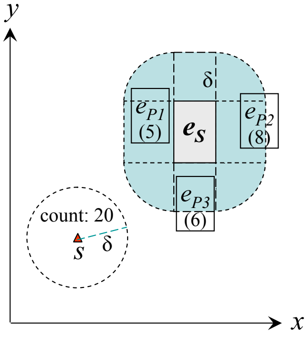


**Fig. 7.** Pruning using $\Phi^*(e_S)$

First, we introduce the concept of $\delta$-Minkowski region [3] of $e_S$, denoted by $\Xi(e_S, \delta)$, which is the set of possible locations whose minimum distance from $e_S$ is within the distance $\delta$.

$$\Xi(e_S, \delta) = \{t \in \mathbb{R}^2 \mid mindist(t, e_S) \leq \delta\} \tag{2}$$

The region $\Xi(e_S, \delta)$ is illustrated in Figure 5 as the region extended from the rectangle $e_S$ by the distance $\delta$. Given the dataset $P$ and the distance $\delta$, we define the *upper bound neighborhood dominating score* $\Phi^+_{P,\delta}(e_S)$ of $e_S$ as the number of objects in $P$ that fall into the region $\Xi(e_S, \delta)$.

$$\Phi^+_{P,\delta}(e_S) = |\{o \in P \mid o \subseteq \Xi(e_S, \delta)\}| \tag{3}$$

The nice property of the upper bound score $\Phi^+_{P,\delta}(e_S)$ (of a non-leaf entry $e_S$) is that it is guaranteed to be greater than or equal to the score $\Phi_{P,\delta}(s)$ of any object $s$ in the subtree of $e_S$. This is shown in the following lemma.

**Lemma 1.** *Let $\delta$ be a distance threshold and $P$ be a dataset of objects. Given a rectangle $e_S$, it holds that $\Phi_{P,\delta}(s) \leq \Phi^+_{P,\delta}(e_S)$ for any object $s$ that falls into $e_S$.*

*Proof.* Let $s$ be an object that falls into $e_S$. According to Definition 1, each object $o \in P$ that contributes to $\Phi_{P,\delta}(s)$ must satisfy the inequality $dist(o, s) \leq \delta$ (and also the condition $o.\psi \prec s.\psi$). Each such object $o$ also satisfies $mindist(o, e_S) \leq \delta$ because $s$ falls into $e_S$. That means such object $o$ falls into the region $\Xi(e_S, \delta)$ and thus contributes to $\Phi^+_{P,\delta}(e_S)$. Therefore, we have $\Phi_{P,\delta}(s) \leq \Phi^+_{P,\delta}(e_S)$. ■

We proceed to present the **EntryScore** algorithm, whose pseudo code is shown in Algorithm 3. It takes as input (i) an entry $e_S$ of the R-tree $R_S$ on $S$, (ii) an entry $e_P$ in the aggregate R-tree $R_P$ on $P$, and (iii) the distance threshold $\delta$. This algorithm serves for two purposes, depending on whether $e_S$ is a leaf entry or not. If $e_S$ is a leaf entry, then the algorithm calls the ObjectScore function to compute the exact score $\Phi(e_S)$ of $e_S$ (lines 2–3).

Otherwise, $e_S$ is a non-leaf entry, lines 4–11 are used to compute the upper bound score $\Phi^+(e_S)$ of $e_S$. We then check whether the region $\Xi(e_S, \delta)$ contains $e_P$. If so, then each object in the subtree of $e_P$ is guaranteed to fall in $\Xi(e_S, \delta)$. Thus, we increment the counter $v$ by the count $e_P.count$, without visiting the subtree of $e_P$. If not, then we need to read the child node of $e_P$. An entry $e'_P$ in the child node is recursively processed if it intersects $\Xi(e_S, \delta)$, i.e., having the potential of contributing to $\Phi^+(e_S)$.

**Algorithm 3. EntryScore**(Entry $e_S$ of the R-tree $R_S$ on $S$, Entry $e_P$ in the aggregate R-tree $R_P$ on $P$, Distance $\delta$)

```
1: $v := 0$
2: **if** $e_S$ is a leaf entry **then**
3:     $v :=$ObjectScore$(e_S, e_P, \delta)$
4: **else**                                            ▷ $e_S$ is a non-leaf entry
5:     **if** $\Xi(e_S, \delta)$ contains $e_P$ **then**
6:         $v := e_P.count$
7:     **else**
8:         read the child node $CN$ pointed to by $e_P$
9:         **for** each child $e'_P$ in $CN$ **do**
10:            **if** $\Xi(e_S, \delta)$ intersects $e'_P$ **then**
11:                $v := v+$EntryScore$(e_S, e'_P, \delta)$
12: **return** $v$
```

Figure 6 illustrates an example of computing the upper bound score $\Phi^+(e_S)$ of a non-leaf entry $e_S$ (of the tree $R_S$), by using the EntryScore algorithm. Here, the aggregate R-tree $R_P$ (of the dataset $P$) only has the non-leaf entries $e_{P1}, e_{P2}, e_{P3}$, whose associated count values are 5, 8, 6 respectively. Since $e_{P1}$ is contained by $\Xi(e_S, \delta)$, its count (5) is added to the upper bound score of $e_S$, without visiting the subtree. As the entries $e_{P2}$ and $e_{P3}$ intersect $\Xi(e_S, \delta)$, their child nodes need to be accessed. Then, the child nodes of $e_{P2}$ and $e_{P3}$ are found to have 4 and 3 objects, respectively, that fall into the region $\Xi(e_S, \delta)$. Therefore, the values 4 and 3 are added to the upper bound score of $e_S$. In summary, we obtain: $\Phi^+(e_S) = 5 + 4 + 3 = 12$.

**Search Algorithm**

Recall that we have studied the notion of $\Phi^+(e_S)$ (for a non-leaf entry $e_S$), and the EntryScore algorithm for computing it. We continue to present a pruning rule for reducing the search space, and then develop two algorithms for solving MEO based on the pruning rule.

According to Lemma 1, we devise the following pruning rule to identify an unpromising entry $e_S$ (of the R-tree $R_S$ on $S$) whose subtree cannot contain the MEO object.

**Pruning Rule 1.** *Let $s \in S$ be an object from $S$, and $e_S$ be a non-leaf entry from R-tree $R_S$ on $S$. If $\Phi(s) > \Phi^+(e_S)$, then the entry $e_S$ can be safely pruned.*

We continue with the example of Figure 6 to illustrate this pruning rule. Suppose that we have already examined object $s$ and computed its exact value $\Phi(s) = 20$ (by the EntryScore algorithm). Next, we want to check whether it is necessary to visit the subtree of the non-leaf entry $e_S$. Its upper bound score $\Phi^+(e_S) = 5+4+3 = 12$ can be computed by the EntryScore algorithm, as discussed before. Since $\Phi^+(e_S) < \Phi(s)$, the entry $e_S$ cannot contribute to the result and therefore it can be safely pruned.

It is desirable to find early an object $s$ with high $\Phi(s)$ value such that unqualified subtrees of $R_S$ can be effectively pruned. The search on $R_S$ can be conducted in two tree search paradigms, namely *best-first search* or *depth-first search*.

The pseudo code of the *best-first search* (BFS) is shown in Algorithm 4. It employs a max-heap $H$ so as to visit the tree entries of $R_S$ in descending order of their upper bound scores. Initially, the algorithm inserts into $H$ the root entry of $R_S$ together with its upper bound score $|P|$. Each time a non-leaf entry is deheaped, all its child entries are enheaped with their own priorities obtained by calling the EntryScore algorithm (lines 5–8). If the entry being deheaped is a leaf entry, it will be returned as the most endangered object (line 9–10). The correctness of the BFS algorithm is guaranteed by (i) the property of the max-heap, and (ii) the upper bound computed by EntryScore($R_P.root, e_S, \delta$) (stated in Lemma 1).

**Algorithm 4. BFS**(Aggregate R-tree $R_P$ on $P$, R-tree $R_S$ of $S$, Distance $\delta$)

```
1: initialize a max-heap $H$
2: enheap($H, \langle R_S.root, |P| \rangle$)
3: while $H$ is not empty do
4:     $e_S$ :=deheap($H$)
5:     if $e_S$ is a non-leaf entry then
6:         read the child node $CN$ pointed to by $e_S$;
7:         for each child $e'_S$ of $e_S$ do
8:             enheap($H, \langle e'_S$, EntryScore($e'_S, R_P.root, \delta$)$\rangle$)
9:     else
10:        return $e_S$
```

Similarly, it is also possible to traverse the R-tree $R_S$ in the depth-first manner. The resulting algorithm is called *the depth-first search* (DFS). Due to the space limit, we omit its pseudo code here.

## 3.3 Spatial Join Based Algorithm

As in Section 3.2, here we assume that the set of candidates objects $S$ is indexed by an R-tree $R_S$ and the set of competitor objects $P$ is indexed by an aggregate

R-tree $R_P$. Recall that both the BFS and DFS algorithms need to compute the upper bound score of a non-leaf entry $e_S$ (of the tree $R_S$) explicitly by accessing the tree $R_P$, incurring considerable cost. This section presents a more efficient solution by deriving an upper bound score of $e_S$ with low cost and tightening the score bound gradually whenever necessary.

**Formulation of a Join List**
Before proposing the solution, we first introduce several relevant concepts. Let $e_S$ be an entry of the R-tree $R_S$. At query time, we associate each encountered entry $e_S$ with its *join list* $e_S.JL$, for storing the entries of the R-tree $R_P$ that may combine with the subtree of $e_S$ to generate potential results.

Specifically, a join list $e_S.JL$ is required to satisfy both of these conditions:

– (i) each entry $e_P$ in $e_S.JL$ satisfies $e_P \cap \Xi(e_S, \delta) \neq \emptyset$,
– (ii) for each $p \in P$ satisfying $p \cap \Xi(e_S, \delta) \neq \emptyset$, there is exactly one ancestor entry $e_P$ (of $p$) in $e_S.JL$.

The first condition ensures that the entries stored in $e_S.JL$ are relevant to $e_S$ because they intersect the Minkowski region $\Xi(e_S, \delta)$ of $e_S$. The second condition ensures that there is no missing entry or redundant entry in $e_S.JL$.

The next question is how to check whether a particular join list satisfies both conditions (i) and (ii) stated above. First of all, we start with the root join list $e_S.JL = \{R_P.root\}$, which trivially satisfies the condition (ii). The condition (i) can be easily checked on $e_S.JL$. In each subsequent step, we can apply the following *expansion operation* on $e_S.JL$; this operation guarantees that its output join list must satisfy both conditions (i) and (ii). Each time, we pick an non-leaf entry $e_P$ from $e_S.JL$, read the child node $CN$ pointed to by $e_P$, and then insert each entry $e'_P \in CN$ satisfying $e'_P \cap \Xi(e_S, \delta) \neq \emptyset$ into the list $e_S.JL$.

Having described the concept of a join list $e_S.JL$, we then define the upper bound score of $e_S$ with respect to $e_S.JL$ as:

$$\Phi^*_{P,\delta}(e_S) = \sum_{e' \in e_S.JL} e'.count \tag{4}$$

The above upper bound score $\Phi^*_{P,\delta}(e_S)$ is guaranteed to be greater than or equal to the score $\Phi_{P,\delta}(s)$ of any object $s$ in the subtree of $e_S$. This is formally stated in the following lemma.

**Lemma 2.** *$\Phi^*_{P,\delta}(e_S) \geq \Phi_{P,\delta}(s)$ for any object $s$ that falls into $e_S$.*

*Proof.* Let $s$ be an object that falls into $e_S$. According to Lemma 1, we obtain $\Phi^+_{P,\delta}(e_s) \geq \Phi_{P,\delta}(s)$. From the property (ii) of the join list, we derive $\Phi^*_{P,\delta}(e_S) \geq \Phi^+_{P,\delta}(e_S)$. By combining both inequalities above, we have $\Phi^*_{P,\delta}(e_S) \geq \Phi_{P,\delta}(s)$. ■

We illustrate an example on exploiting the upper bound score of join list for pruning unnecessary subtrees of the tree $R_S$. Figure 7 shows a non-leaf entry $e_S$. Suppose that we have encountered an object with $\Phi_{P,\delta}(s) = 20$. Next, we check whether it is necessary to access the child node of $e_S$. Suppose that

$e_S.JL = \{e_{p1}, e_{p2}, e_{p3}\}$, and $\Phi^*_{P,\delta}(e_S) = 5 + 8 + 6 = 19 < 20$, i.e., lower than the score of object $s$. Therefore, the entry $e_S$ (together with its join list) can be safely pruned as no object in $e_S$ can have higher score than $s$.

**Search Algorithm**

Algorithm 5 is the pseudo code of the spatial join based algorithm. It employs a max-heap $H$ to keep all $R_S$ entries to be processed. Each $R_S$ entry $e_S$ is enheaped together with its join list $e_S.JL$ and a count value obtained from Equation 4.

If the $R_S$ entry $e_S$ being deheaped is a leaf node and its join list is null, it is returned as the most endangered object according to the max-heap property (lines 5–8). If the leaf entry $e_S$'s join list is not null, its exact neighbor dominator count is calculated by calling the ObjectScore algorithm for each entry in its join list (lines 10–12). After that, $e_S$ is enheaped again with a null join list and the calculated count value (line 13).

**Algorithm 5. SJB**(Aggregate R-tree $R_P$ of $P$, R-tree $R_S$ of $S$, Distance $\delta$)

```
 1: initialize a max-heap $H$
 2: $e_{root} := R_S.root$; $e_{root}.JL := \{R_P.root\}$
 3: enheap($H, \langle e_{root}, e_{root}.JL, 0\rangle$)
 4: **while** $H$ is not empty **do**
 5:     $\langle e_S, e_S.JL\rangle$ := deheap($H$)
 6:     **if** $e_S$ is a leaf entry **then**
 7:         **if** $e_S.JL$ is null **then**
 8:             **return** $e_S$
 9:         **else**
10:             $v := 0$
11:             **for** each $e_j$ in $e_S.JL$ **do**
12:                 $v := v+$ObjectScore($e_S, e_j, \delta$)
13:             enheap($H, \langle e_S, null, v\rangle$)
14:     **else**
15:         read the child node $CN_S$ pointed to by $e_S$
16:         **for** each entry $e_i$ in $CN_S$ **do**
17:             $v := 0$; $e_i.JL := \emptyset$
18:             **for** each $e_j$ in $e_S.JL$ **do**
19:                 **if** $\Xi(e_i, \delta)$ contains $e_j$ **then**
20:                     add $e_j$ to $e_i.JL$;  $v := v + e_j.count$
21:                 **else**
22:                     read the child node $CN_P$ pointed to by $e_j$
23:                     **for** each child $e'$ in $CN_P$ **do**
24:                         **if** $\Xi(e_i, \delta)$ intersects $e'$ **then**
25:                             add $e'$ to $e_i.JL$;  $v := v + e'.count$
26:             enheap($H, \langle e_i, e_i.JL, v\rangle$)
```

Otherwise, the join is executed by expanding the non-leaf $R_S$ entry $e_S$ being deheaped, and enheaping each subentry in $e_S$ with its corresponding join list and count value (15–26). In particular, when $e_S$ is expanded its each subentry $e_i$ gets part of entries in $e_S.JL$ as $e_i.JL$. In this way, as the join proceeds on the

$R_S$ entries are enheaped with join lists of smaller coverage, thus giving tighter upper bounds of neighbor dominator counts which favors pruning.

## 4 Experimental Study

In this section, we experimentally evaluate our proposed algorithms for processing MEO queries. All algorithms were implemented in Java and were run on a Windows XP PC with a 2.8GHz Intel Pentium D CPU and 1GB RAM. We used synthetic and real datasets for both the competitor set $P$ and the candidate set $S$. In each dataset, all spatial coordinates are normalized to Euclidean space $[0,10000]^2$, whereas each quality attribute is normalized to the unit interval [0,1]. In all experiments, the disk page size is set to 4KBytes and a LRU memory buffer with 512KBytes is used. We issue 20 queries for each test case in each experiment, and the spatial distance constraint $\delta$ in each query is a random value in (0,1000]. We measure the average node access cost per each query because it dominates the total query processing cost.

### 4.1 Experimental Results on Real Data

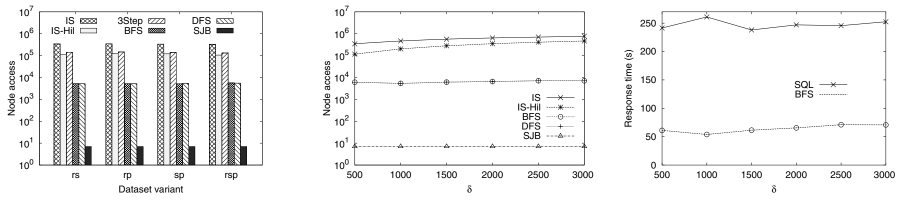
In this section, we used two real datasets from AllStays.com[1] that maintains collections of hotels, resorts, campgrounds, etc. around the world. We chose the dataset of hotels in US and cleaned it up as follows. We removed all those records without longitude and latitude, and discarded quality attributes with very few non-null values. For each remaining quality attribute, any null value was replaced by a random value from its attribute domain. As a result, we obtained 30,918 hotel records with the schema (*longitude*, *latitude*, *review*, *stars*, *price*). Value conversion was done on a quality attribute if necessary, e.g., a higher stars value was converted to a lower value in the normalized range [0, 1]. This way, lower values are preferable to higher ones. After normalizing the hotel records as mentioned above, one third (10,306 records) are randomly picked as the $S$ dataset and the others (20,612 records) form the $P$ dataset.

We then used different quality attribute combinations and got four variants of $P$ dataset: *review* and *stars* (denoted as rs), *review* and *price* (denoted as rp), *stars* and *price* (denoted as sp), and all three attributes (denoted as rsp). The corresponding $S$ dataset variants were obtained in the same way. We then performed three groups of experiments on the real datasets obtained.

The first group of experiments investigated into the effect of different real dataset variants. We also implemented the multi-step R-tree based solution mentioned in Section 2.3 (named 3Step for short). The average node access cost results are reported in Figure 8(a). The 3Step solution is inefficient because it cannot exploit the characteristics of the MEO query for effective pruning. The SJB algorithm has the lowest cost.

The second group of experiments studied the effect of distance constraint $\delta$. We used the rsp datasets and varied $\delta$ from 500 to 3000. The results are reported

[1] Hotel and Travel Guide. http://www.allstays.com/



(a) Node access, all datasets (b) Node access vs. $\delta$, on rsp (c) Response time, on rsp

**Fig. 8.** Results on real datasets

in Figure 8(b). Since SJB applies an effective pruning technique, it outperforms the other algorithms.

In the last group of experiments, we executed the RDBMS solution mentioned in Section 2.3 (named SQL for short) on the rsp dataset. We executed SQL in the Oracle RDBMS; we could only obtain the response time of SQL but not its I/O cost. Therefore, we compared the response time of SQL with our proposed BFS, DFS and SJB algorithms (see the results in Figure 8(c)). Obviously, our algorithms incur considerably shorter response time than SQL.

In subsequent experiments, we discard the solutions 3Step and SQL due to their high query cost.

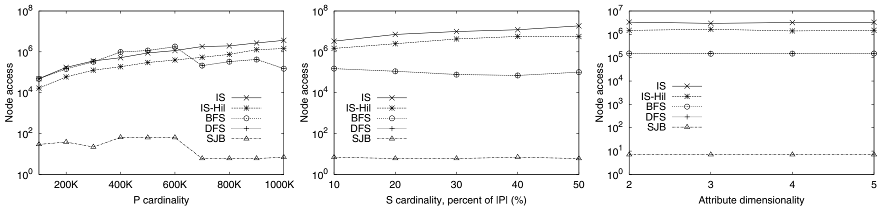
## 4.2 Scalability and Robustness Experiments on Synthetic Data

Having studied the performance of our algorithms on real data, we now test their scalability and robustness by using synthetic data. Table 2 lists the parameters for the generation of synthetic datasets; the default parameter values are shown in bold. The cardinality of the set $P$ varies from 100K to 1000K. For each $P$ set, the cardinality of the corresponding $S$ set changes from 10% to 50% of the cardinality of $P$. For both $P$ sets and $S$ sets, all locations are generated randomly in the normalized Euclidean space $[0,10000]^2$. The quality attribute dimensionality of those datasets varies from 2 to 5. For both $P$ and $S$ sets, we generated quality values following the independent (IN) distribution and the anti-correlated (AC) distribution, according to Borzonyi et al. [1].

**Table 2.** Parameters of synthetic datasets

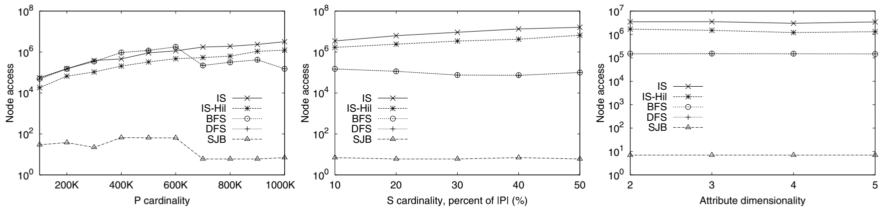
| **Parameter** | **Setting** |
|---|---|
| Competitor dataset cardinality, $\|P\|$ | 100K, 200K, . . ., **1000K** |
| Candidate dataset cardinality, $\|S\|$ | **10%**$\cdot\|P\|$, 20%$\cdot\|P\|$, . . ., 60%$\cdot\|P\|$ |
| Quality attribute dimensionality, $c$ | **2**, 3, 4, 5 |
| Quality attribute distribution | Independent (IN), Anti-correlated (AC) |

**Effect of Competitor Dataset Cardinality** $|P|$
We first varied $|P|$ from 100K to 1000K to see its effect on the performance of all algorithms. The results are reported in Figure 9(a) and Figure 10(a), on IN and AC distribution respectively. As $|P|$ increases, both IS and IS-Hil algorithms



(a) Effect of $|P|$ (b) Effect of $|S|$ (c) Effect of $c$

**Fig. 9.** Node access on synthetic datasets, with IN quality attributes



(a) Effect of $|P|$ (b) Effect of $|S|$ (c) Effect of $c$

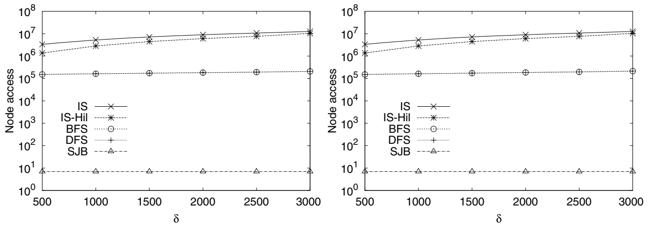
**Fig. 10.** Node access on synthetic datasets, with AC quality attributes

incur more node accesses. The reason is that the recursive ObjectScore algorithm needs to access more nodes from the R-tree on a larger $P$. While BFS and DFS algorithms degrade first and then improve as $|P|$ increases. As $|S|$ was fixed to 10% of that of $|P|$, a larger $P$ leads to a larger $S$ and a larger R-tree on $S$. Therefore BFS and DFS degrade as they have to search a larger R-tree. The subsequent improvement is resulted from a better organized R-tree on $S$, which is achieved only when $S$ contains enough objects. The performance of the SJB algorithm fluctuates more visibly, as the join operation complicates the use of both the R-tree on $S$ and the aggregate R-tree on $P$. Among all algorithms, SJB performs the best as the join is very efficient due to the powerful pruning rule it employs.

**Effect of Candidate Dataset Cardinality $|S|$**
We then varied $|S|$ from 10% to 50% of that of $|P|$. The results on the effect of varying $|S|$ are reported in Figure 9(b) and Figure 10(b). Observe that the SJB algorithm still outperforms all others; BFS and DFS are the second best among all. As $|S|$ increases, both BFS and DFS improve slightly. This again is attributed to a better organized R-tree on $S$.

Both IS and IS-Hil algorithms degrades slightly as $|S|$ increases, because they invoke the recursive ObjectScore algorithm for every object in $S$. Whereas the latter performs slightly better steadily as $|S|$ varies. Accessing objects of $S$ in the Hilbert curve order enables IS-Hil to considerably reuse $R_P$ tree nodes in the buffer, which offsets the effect of increasing $|S|$.

**Effect of Quality Attribute Dimensionality $c$**
To observe the effect of quality attribute dimensionality, we varied the quality dimensionality $c$ from 2 to 5. The results on IN and AC data are shown



(a) IN quality attributes (b) AC quality attributes

**Fig. 11.** Node access vs. $\delta$

in Figure 9(c) and Figure 10(c) respectively. All algorithms are insensitive to the variation of attribute dimensionality. This is attributed to the fact that all indexes used in those algorithms are spatial access methods. They only index spatial coordinates of all objects, and therefore are not affected much by the number of quality attributes in each object dataset. Here SJB remains to be the best solution.

**Effect of Distance Constraint $\delta$**
We also investigated into the effect of the distance constraint $\delta$ used in most endangered object queries. In this batch of experiments, we used the default settings and changed $\delta$ from 500 to 3000. Figure 11 reports the relevant results. On both attribute distributions, larger distance ranges result in marked performance degradation for IS and IS-Hil algorithms. This is because the ObjectScore algorithm (Algorithm 1) is called more often by these two algorithms when the distance range is larger. In contrast, almost no performance change is visible for the BFS, DFS and SJB algorithms. This implies that the variation of distance range does not affect the pruning power via the the R-trees exploited by those algorithms. Note that SJB still performs the best, indicating the strong pruning power of the upper bound score obtained from an entry's join list.

## 5 Conclusion and Future Work

In this paper we formalize a novel query, the most endangered object query (MEO), which takes into account both spatial distance constraint and multiple quality attributes. Given a competitor object set $P$ and a candidate object set $S$, and a distance $\delta$, the MEO query returns from $S$ an object $s$ such that it maximizes the number of objects of $P$ dominating $s$ within the $\delta$-neighborhood (of $s$). It has important applications in business planning, online war games, wild animal protection, etc.

We propose several algorithms for processing MEO queries efficiently. The IS algorithm is an iterative search approach which requires that only $P$ is indexed by an R-tree. To improve the performance, we index $S$ by an aggregate R-tree, which enables effective pruning by using the aggregate count in each node entry. Then, best-first search (BFS) and depth-first search (DFS) for evaluating the query are studied. A spatial-join based algorithm (SJB) is also developed to process query fast. An extensive experiment study for the above methods is conducted on both synthetic and real datasets. Empirical results show that the SJB algorithm outperforms other solutions and it scales well for large datasets.

Several interesting directions exist for future research. First, we want to capture the realistic scenario that dominators (from $P$) tend to have more impact on a candidate object $s \in S$ when they are close to $s$. For this, the score function (in Definition 1) can be redefined by assigning higher weights to competitors that are close to $s$. The challenge is then to extend our proposed algorithms for such a weighted score function. Second, the set $S$ of candidate locations can appear in other forms than a location set. For example, certain practical applications need to find the most endangered location(s) on a pre-defined trajectory of a wild animal or a military operation. Third, it is also of interest to define a generic query type that combines spatial locations and quality attributes. A fundamental solution can be developed for such a generic definition, which can be instantiated to process concrete queries like MEO queries in this paper.

*Acknowledgments.* We thank the anonymous reviewers and SSDBM 2009 PC Chair for their constructive comments.

## References

1. Borzonyi, S., Kossmann, D., Stocker, K.: The Skyline Operator. In: Proc. of ICDE, pp. 421–430 (2001)
2. Brinkhoff, T., Kriegel, H.-P., Seeger, B.: Efficient Processing of Spatial Joins Using R-Trees. In: Proc. of SIGMOD, pp. 237–246 (1993)
3. Böhm, C.: A Cost Model for Query Processing in High Dimensional Data Spaces. ACM TODS 25(2), 129–178 (2000)
4. Butz, A.R.: Alternative Algorithm for Hilbert's Space-Filling Curve. IEEE TOC C-20(4), 424–426 (1971)
5. Du, Y., Zhang, D., Xia, T.: The Optimal-Location Query. In: Bauzer Medeiros, C., Egenhofer, M.J., Bertino, E. (eds.) SSTD 2005. LNCS, vol. 3633, pp. 163–180. Springer, Heidelberg (2005)
6. Huang, X., Jensen, C.S.: In-Route Skyline Querying for Location-Based Services. In: Kwon, Y.-J., Bouju, A., Claramunt, C. (eds.) W2GIS 2004. LNCS, vol. 3428, pp. 120–135. Springer, Heidelberg (2005)
7. Huang, Z., Lu, H., Ooi, B.C., Tung, A.K.H.: Continuous Skyline Queries for Moving Objects. IEEE TKDE 18(12), 1645–1658 (2006)
8. Koudas, N., Sevcik, K.C.: High Dimensional Similarity Joins: Algorithms and Performance Evaluation. In: Proc. of ICDE, pp. 466–475 (1998)
9. Li, C., Ooi, B.C., Tung, A.K.H., Wang, S.: DADA: A Data Cube for Dominant Relationship Analysis. In: Proc. of SIGMOD, pp. 659–670 (2006)
10. Li, C., Tung, A.K.H., Jin, W., Ester, M.: On Dominating Your Neighborhood Profitably. In: Proc. of VLDB, pp. 818–829 (2007)
11. Moon, B., Jagadish, H.V., Faloutsos, C., Saltz, J.H.: Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. IEEE TKDE 13(1), 124–141 (2001)
12. Papadias, D., Kalnis, P., Zhang, J., Tao, Y.: Efficient OLAP Operations in Spatial Data Warehouses. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) SSTD 2001. LNCS, vol. 2121, pp. 443–459. Springer, Heidelberg (2001)

13. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An Optimal and Progressive Algorithm for Skyline Queries. In: Proc. of SIGMOD, pp. 467–478 (2003)
14. Sharifzadeh, M., Shahabi, C.: The Spatial Skyline Queries. In: Proc. of VLDB, pp. 751–762 (2006)
15. Xia, T., Zhang, D., Kanoulas, E., Du, Y.: On Computing Top-t Most Influential Spatial Sites. In: Proc. of VLDB, pp. 946–957 (2005)
16. Yiu, M.L., Dai, X., Mamoulis, N., Vaitis, M.: Top-k Spatial Preference Queries. In: Proc. of ICDE, pp. 1076–1085 (2007)
17. Zhang, D., Du, Y., Xia, T., Tao, Y.: Progressive Computation of the Min-dist Optimal-Location Query. In: Proc. of VLDB, pp. 643–654 (2006)
18. Zheng, B., Lee, K.C.K., Lee, W.-C.: Location-Dependent Skyline Query. In: Proc. of MDM, pp. 148–155 (2008)
19. Zhu, M., Papadias, D., Zhang, J., Lee, D.L.: Top-k Spatial Joins. IEEE TKDE 17(4), 567–579 (2005)

# Constraint-Based Learning of Distance Functions for Object Trajectories

Wei Yu[1] and Michael Gertz[2]

[1] Department of Computer Science, University of California, Davis, U.S.A.
weyu@ucdavis.edu
[2] Institute of Computer Science, University of Heidelberg, Germany
gertz@informatik.uni-heidelberg.de

**Abstract.** With the drastic increase of object trajectory data, the analysis and exploration of trajectories has become a major research focus with many applications. In particular, several approaches have been proposed in the context of similarity-based trajectory retrieval. While these approaches try to be comprehensive by considering the different properties of object trajectories at different degrees, the distance functions are always pre-defined and therefore do not support different views on what users consider (dis)similar trajectories in a particular domain.

In this paper, we introduce a novel approach to learning distance functions in support of similarity-based retrieval of multi-dimensional object trajectories. Our approach is more generic than existing approaches in that distance functions are determined based on constraints, which specify what object trajectory pairs the user considers similar or dissimilar. Thus, using a single approach, different distance functions can be determined for different users views. We present two learning techniques, *transformed Euclidean distance* and *transformed Dynamic Time Warping*. Both techniques determine a linear transformation of the attributes of multi-dimensional trajectories, based on the constraints specified by the user. We demonstrate the flexibility and efficiency of our approach with applications to clustering and classification on real and synthetic object trajectory datasets from different application domains.

## 1 Introduction

Driven by major advancements in sensor technology, GPS-enabled mobile devices, and object tracking, large amounts of data describing moving object trajectories are currently generated and managed in various application domains (see, e.g., [17,19] for some excellent surveys). In order to effectively analyze and explore such data for patterns of interest and unexpected phenomena, several data mining techniques for object trajectories have been developed. In most of the applications, trajectory data are typically multi-dimensional. In particular, trajectory data can be treated as multi-dimensional time series and thus respective data analysis techniques can be applied.

A fundamental ingredient of most of the trajectory analysis tasks are distance measures that allow to effectively determine the similarity of trajectories. Respective tasks include trajectory clustering, classification, and k-nearest neighbor search. Several approaches and distance measures have been proposed, tailored to multi-dimensional object trajectories, e.g., [3,20,21]. Common to all the existing approaches is that distances measures (or functions) are explicitly given as part of the framework. Although these measures try to be comprehensive by considering the various dimensions of trajectory data, they typically assume that the dimensions determining similarity are the original dimensions that describe the trajectories. However, there are several application domains where different users have different views on what trajectories are similar and which ones are not. For example, if variables $x$ and $y$ describe 2-d trajectories, for the first user, dimension $x$ might be important in determining similarity while for the second user, $y$ is more important. And a third user considers a linear combination of $x$ and $y$ as a critical factor to describe trajectory similarity. If $x$ is a basic component of the distance function, the similarity measure is of little help to the second and third user. In this sense, existing approaches relying on unsupervised, "hard-coded" distance measures do not provide much flexibility in terms of supporting different user views and domain specific knowledge about trajectories.

In this paper, we address this problem by introducing a novel approach in which, given a set of multi-dimensional object trajectories, distance measures are *learned from constraints*. These constraints are specified by the user and simply consist of a few pairs of trajectories the user considers similar or dissimilar. We propose learning techniques of two distance measures, *transformed Euclidean Distance* and *transformed Dynamic Time Warping*, which are based on posing an optimization problem. Both techniques detect critical attributes that determine similarity between the multi-dimensional trajectories based on the constraints specified by the user. A resulting linear transformation matrix is then simply embedded in a distance function that satisfies the user constraints best. In our approach we do not rely on the traditional techniques that utilize a comprehensive set of labeled training data, as they typically appear in (supervised) classification problems. The types of constraints considered in this paper are much simpler. Consequently, our approach cannot only be applied to traditional classification problems but also to clustering and similarity search for object trajectories. In order to also accommodate similarity views that consider object movement patters at different rates, an important consideration is that the distance measure has the ability to allow time warping along the time axis. We achieve this through our novel transformed Dynamic Time Warping technique, which is different from techniques that simply learn a Mahalanobis distance for multi-dimensional data points in a supervised fashion (e.g., [23]).

Thus, the main contribution of our work is to provide a single framework that allows to derive distance measures that satisfy different user needs and (subjective) views on some given trajectory data. This approach enables the efficient computation of distance measures applicable to various mining tasks for object trajectories, as we will demonstrate in our comprehensive experiments where we use different datasets and consider typical data mining tasks such as trajectory clustering and classification.

The rest of the paper is organized as follows. After a discussion of related work in the following section, in Sec. 3, we outline the concepts underlying trajectories and distance measures. In Sec. 4, we discuss in detail our approach to learning distance functions in the context of user constraints. After a demonstration of the effectiveness of the proposed approach using different datasets in Sec. 5, we conclude the paper in Sec. 6.

## 2 Related Work

Our approach is mostly related to the areas of similarity search for moving object trajectories and distance learning.

Recently, there has been very active research on mining trajectories, which typically can be viewed as (multi-dimensional) time series, and trajectory similarity search. For this, several alternative distance measures have been proposed. In [12], Lee et al. use the Euclidean Distance (ED) for multi-dimensional time series. Euclidean Distance, however, is known to be sensitive to local distortions in the time axis. To address this problem, Vlachos et al. extend Dynamic Time Warping (DTW) and Longest Common Subsequence (LCSS) for multi-dimensional trajectories [20]. Chen et al. in [3] study Edit Distance on Real sequence (EDR), which is based on string edit distance. Chen et al. in [2] propose ERP, which tries to combine the advantages of DTW and EDR. In [24], Wu et al. study a One-way distance (OWD) function to compute the spatial similarity of trajectories. Lee et al. also developed approaches for trajectory clustering [11] based on sub-trajectories. Vlachos et al. present a DTW-based distance measure that is invariant to rotation in [21].

All these approaches are designed for multi-dimensional trajectories. However, all these proposed distance measures assume that the dimensions used to describe trajectory similarity are the original dimensions that describe the trajectories. Our work is different from these approaches, because in our approach, distance measures are learned from user-specified constraints and are thus able to satisfy user views on trajectory similarity. Users are allowed to give examples to indicate which trajectories they consider similar and/or dissimilar. Our algorithm then detects meaningful underlying dimensions that allow to explain the similarity and/or dissimilarity, and yields a distance measure that satisfies the user constraints best.

Our work is also different from traditional unsupervised feature extraction techniques such as principal component analysis (PCA), factor analysis (FA), and Isomap. Also these techniques do not guarantee to capture the trajectory properties that are of interest to the user. They are learned only using the intrinsic properties of the trajectory data. For example, PCA simply performs a coordinate rotation that aligns the transformed axes with the directions of maximum variance.

There also has been considerable work on supervised distance learning, such as learning a distance function for classification problems (e.g., [5,25]). However, a problem with these methods is that they need a labeled training dataset for which then the classification accuracy can be optimized. We do not assume such a labeled training dataset but only a few simple user constraints that indicate which trajectories are (dis)similar. Xing et al. in [23] assume similar user constraints and propose a global Mahalanobis distance for data points that respect such constraints. However, we believe that their method does not solve our problem at hand, for the following reasons. First, the objective of our approach is to learn a good distance function for multi-dimensional object trajectories. Their approach, including many other distance learning techniques (see [25]), try to learn a distance for multi-dimensional data points, where only a snapshot of attribute values is meaningful. Second, many trajectory mining tasks aim to discover objects that have similar movement patterns, even at different rates. Hence, the distance measure is expected to have the ability to allow time warping in the time axis. This important consideration, however, is not present in learning a distance measure for multi-dimensional data points.

## 3 Background: Trajectories and Distances

In this paper, we consider object trajectories where $n$ measurements are recorded at discrete points in time. For a trajectory of the pattern $P = [p_1, p_2, \ldots, p_m]$, each component $p_i$ is an $n$-dimensional vector of measurements (attributes) $(p_{1,i}, p_{2,i}, \ldots, p_{n,i}) \in \mathbb{R}^n$ recorded at time $i$. We refer to the number of time instants in $P$ as *size* of the trajectory.

For one-dimensional time series (trajectory data), the Euclidean distance (ED) and Dynamic Time Warping (DTW) distance are commonly used as similarity measures (e.g., [1,7,15,16]). Both distance measures can easily be extended to multi-dimensional trajectories as suggested, for example, by Vlachos et al. [20]. Because Euclidean distance is only defined for trajectories of the same size, an interpolation needs to be applied to the input trajectories. For this, different techniques can be used (see, e.g., [9,15]).

**Definition 1.** *Given two $n$-dimensional trajectories $P$ and $Q$ of size $k$ (after interpolation). The Euclidean Distance between $P$ and $Q$, denoted $ED(P, Q)$, is defined as*

$$ED(P, Q) := \sqrt{\sum_{i=1}^{k} (p_i - q_i)^T (p_i - q_i)}$$

For the DTW distance, input trajectories do not necessarily have to have the same size. In the following, for a trajectory $P = [p_1, p_2, \ldots, p_m]$, we denote with $P_{1\ldots i}$ the sub-trajectory of $P$ containing the $i$ first elements of $P$, i.e., $P_{1\ldots i} = [p_1, p_2, \ldots, p_i]$.

**Definition 2.** *Given two $n$-dimensional trajectories $P$ and $Q$ of size $m$ and $l$, respectively. The DTW distance between $P$ and $Q$, denoted as $DTW(P, Q)$, is determined by evaluating the recurrence equation*

$$DTW(P_{1\ldots i}, Q_{1\ldots j}) := \sqrt{(p_i - q_j)^T(p_i - q_j) + \min \begin{cases} DTW^2(P_{1,\ldots,i-1}, Q_{1,\ldots,j-1}) \\ DTW^2(P_{1,\ldots,i-1}, Q_{1,\ldots,j}) \\ DTW^2(P_{1\ldots,i}, Q_{1,\ldots,j-1}) \end{cases}}$$

Note that some papers (e.g., [20,22]) omit the square root function in the definition of DTW. It is only used for some optimizations and does not change the essence of the function [22]. DTW finds the optimal alignment between two time series so that their distance is minimized. An alignment can be obtained by using dynamic programming to solve the recurrence equation. In order to speed up the computation and to prevent pathological warping, band constraints such as the Sakoe-Chiba band [18] can be applied. For more details on DTW, we refer the reader to [1,15].

## 4 Learning Distance Functions

In this section, we introduce two flexible distance functions for multi-dimensional object trajectories, called *transformed Euclidean distance* and *transformed DTW*. We present the learning algorithms underlying these two distance functions in Sections 4.1 and 4.2, respectively.

Throughout the paper, we use the notion of *(user) constraints*. For this, assume a given set of trajectories $S = \{s_i\}_{i=1}^u$, $s_i \in R^{n \times m}$. Constraints are specified by the user in the form of two sets: the $ML$ (*Must-Link*) set and the $CL$ (*Cannot-Link*) set. If the user specifies a pair $(s_i, s_j)$ to be in $ML$, then he (subjectively) considers the trajectories $s_i$ and $s_j$ to be similar. Analogously, he specifies a pair $(s_i, s_j)$ to be in $CL$, if he considers $s_i$ and $s_j$ to be dissimilar.

Given $ML$ and $CL$ constraints, the objective is now to learn a distance measure for a given set of trajectories such that the pairs of trajectories in $ML$ end up to be similar to each other (based on the computed distance) and the pairs in $CL$ are dissimilar to each other. For this, we first propose the following distance functions, transformed Euclidean distance and transformed DTW, to be learned. After this, we provide the intuition behind these two functions.

**Definition 3.** *Given two n-dimensional trajectories $P$ and $Q$ of length $k$ after interpolation and a real symmetric positive semi-definite matrix $A \in R^{n \times n}$. The* Transformed Euclidean distance *between $P$ and $Q$, denoted $ED_A(P, Q)$, is defined as*

$$ED_A(P, Q) := \sqrt{\sum_{i=1}^{k} (p_i - q_i)^T A (p_i - q_i)}.$$

**Definition 4.** *Given two n-dimensional trajectories $P$ and $Q$ of length $m$ and $l$, respectively, and a real symmetric positive semi-definite matrix $A \in R^{n \times n}$. The* Transformed Dynamic Time Warping distance *between $P$ and $Q$, denoted $DTW_A(P, Q)$, is obtained by evaluating the following recurrence equation*

$$DTW_A(P_{1\dots i}, Q_{1\dots j}) := \sqrt{(p_i - q_j)^T A(p_i - q_j) + \min \begin{cases} DTW_A^2(P_{1,\dots,i-1}, Q_{1,\dots,j-1}) \\ DTW_A^2(P_{1,\dots,i-1}, Q_{1,\dots,j}) \\ DTW_A^2(P_{1\dots i}, Q_{1,\dots,j-1})) \end{cases}}$$

In order to give the intuition behind the two distance measures and explain what $A$ actually is, we introduce the concept of *transformed trajectories.*

**Definition 5.** *Given a trajectory* $P = [p_1, p_2, \dots, p_m], p_i \in R^n$ *and a linear transformation matrix* $W \in R^{n \times n}$. *The* transformed trajectory*, denoted* $P_W$*, is defined as* $P_W := [Wp_1, Wp_2, \dots, Wp_m]$.

The transformation of $P$ using matrix $W$ thus can be considered as applying a linear transformation matrix $W$ to each point $p_i (1 \le i \le m)$ in $P$, replacing $p_i$ with $Wp_i$. If and only if $A$ is real symmetric, positive semi-definite, one can find a real matrix $W = A^{1/2}$, i.e. $A = W^T W$. Then one can also find a correspondence between $\mathrm{ED}_A$, $\mathrm{DTW}_A$, and the transformed trajectories, as discussed below. Given above definitions, it is straightforward to see that

(1) $\mathrm{ED}_A(P, Q) = \mathrm{ED}(P_{A^{1/2}}, Q_{A^{1/2}})$ and
(2) $\mathrm{DTW}_A(P, Q) = \mathrm{DTW}(P_{A^{1/2}}, Q_{A^{1/2}})$

Hence, learning $\mathrm{ED}_A(P, Q)$ or $\mathrm{DTW}_A(P, Q)$ is equivalent to finding a transformation matrix $W (W = A^{1/2})$ for all trajectories such that all pairs of trajectories in $ML$ end up to be similar to each other, and the pairs of trajectories in $CL$ are dissimilar to each other.

## 4.1 Learning a Transformed Euclidean Distance from Constraints

In the following, we present the algorithm for learning the transformed Euclidean distance $\mathrm{ED}_A$ for a set of trajectories from some user-specified $ML$ and $CL$ constraints. We pose the distance learning approach as an optimization problem in a way similar to the approach proposed by Xing el al. in [23]. Their approach focuses on learning a distance metric for multi-dimensional data points, while our method tries to learn distance functions for multi-dimensional object trajectories. Intuitively, the desired transformed Euclidean distance measure should bring each pair of trajectories $(s_i, s_j)$ in $ML$ as close as possible, and each pair of trajectories $(s_i, s_j)$ in $CL$ as far apart as possible. Therefore, for the optimization, the sum of squared distances between all pairs $(s_i, s_j)$ in $ML$, denoted $f(A)$, is to be minimized, with the constraint that the sum of distances between all pairs $(s_i, s_j)$ in $CL$, denoted $g(A)$, is greater than the constant 1 to ensure that $A \neq 0$. In the constraint formulation (2) below, we do not use $ED_A^2(s_i, s_j)$ because this then always leads to the matrix $A$ having rank 1. As indicated earlier, another constraint is that the matrix $A$ is positive semi-definite, denoted $A \ge 0$. The optimization problem then is stated as follows:

$$\min_A \ f(A) = \sum_{(s_i, s_j) \in ML} ED_A^2(s_i, s_j) \tag{1}$$

$$\text{s.t.} \quad g(A) = \sum_{(s_i,s_j)\in CL} ED_A(s_i,s_j) \geq 1 \text{ and } A \geq 0 \tag{2}$$

Note that the constant 1 on the right hand side of the first constraint in (2) can be replaced by any other positive constant $c$ because this only leads to $A$ being replaced by $c^2A$. According to the definition of $\mathrm{ED}_A$, the objective function $f(A)$ and both constraints are convex functions. Therefore, one can efficiently find the global optimum. If $A$ is a diagonal matrix, then the transformation $W$ (i.e., $A^{1/2}$) is a scaling matrix. Each diagonal entry $W(i,i)$ can be considered as the "weight" assigned to the dimension $d_i, 1 \leq i \leq n$. Finding the optimum is then equivalent to minimizing

$$\sum_{(s_i,s_j)\in ML} ED_A^2(s_i,s_j) - \log\left(\sum_{(s_i,s_j)\in CL} ED_A(s_i,s_j)\right)$$

To solve this optimization problem, we choose the well-known Newton-Raphson method as in [23]. If $A$ is a full $n \times n$ square matrix, this method takes $O(n^6)$ to invert the Hessian matrix. For efficiency, we solve the problem using gradient ascent and iterative projections in $O(n^2)$, as proposed by Xing et al. in [23].

### 4.2 Learning Transformed Dynamic Time Warping from Constraints

We now discuss the learning algorithm for $\mathrm{DTW}_A$, which can address the problem of distortions in the time axis. First, we pose the optimization problem. Then, we discuss the case where $A$ is diagonal and finally present the solution of a full matrix $A$.

**Optimization Problem.** Based on an idea similar to deriving the optimization problem in Sec. 4.1, we now simply replace the distance function $\mathrm{ED}_A$ with $\mathrm{DTW}_A$ and obtain our new optimization problem, denoted OP1, as follows:

$$\min_A \quad f(A) = \sum_{(s_i,s_j)\in ML} DTW_A^2(s_i,s_j) \tag{3}$$

$$\text{s.t.} \quad g(A) = \sum_{(s_i,s_j)\in CL} DTW_A(s_i,s_j) \geq 1 \text{ and } A \geq 0 \tag{4}$$

However, according to the definition of $\mathrm{DTW}_A$, the objective function $f(A)$ and the constraints are not convex, and it thus cannot be solved using the methods presented earlier. We now formulate the problem in a different way, denoted OP2, as follows:

$$\min_A \quad h(A) = \frac{f(A)}{g^2(A)} = \frac{\sum_{(s_i,s_j)\in ML} DTW_A^2(s_i,s_j)}{(\sum_{(s_i,s_j)\in CL} DTW_A(s_i,s_j))^2} \tag{5}$$

$$\text{s.t.} \quad trace(A) = 1 \text{ and } A \geq 0 \tag{6}$$

OP2 is in fact equivalent to OP1. Due to space constraints, we only give proof sketch here: Suppose $A_{OP1}$ is a solution of OP1, we can prove that $A_{OP1}$ multiplied by a certain constant is a solution of OP2, and vise versa. According to the definition of $DTW_A$, a constant factor to $A$ only results in all pairwise $DTW_A$ distances between trajectories multiplied by the square root of this constant. Thus it does not change the clustering, classification, or similarity search result.

In OP2, the objective function $h(A)$ is derived from $f(A)$ and $g(A)$. $h(A)$ is to be minimized under the constraints that the trace of $A$ is 1 and $A$ is positive semi-definite ($A \geq 0$). Our motivation for this specific choice is that it significantly simplifies the heuristic search. The solution $A$ is required to satisfy all the constraints and minimize the objective function. If we treat the set of matrices satisfying the constraints as the search space, then the task is to find a matrix within this search space that minimizes the objective function. In the problem formulation OP2, the search space is the set of all positive semi-definite matrices with trace 1, which is equivalent to the set of symmetric matrices whose eigenvalues add up to 1 and each eigenvalue is non-negative. As we will show in the next sections, this search space is constructed very easily.

**The case of a diagonal matrix *A*.** In the following, we consider the case of learning a diagonal matrix $A$, which corresponds to a scaling matrix $W$ whose diagonal entries $W(i,i)$ are considered the “weight” assigned to dimension $i$.

In this approach, we use a hill-climbing search algorithm to find a good solution for $A$. The components of the search algorithm are the initial state, heuristic function, successor function, and goal test. We first give the intuition behind this approach before explaining how we realize these four components.

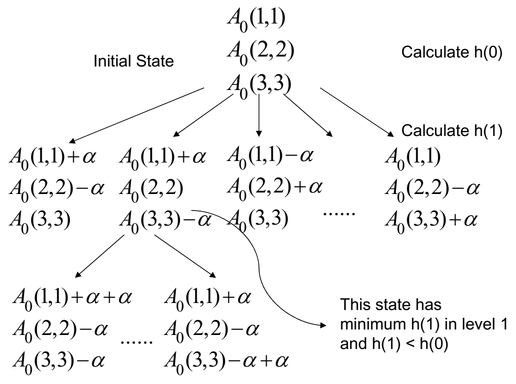
We denote the diagonal matrix $A$ from learning $\text{ED}_A$ in Sec. 4.1 as $A_0$, that is, $A_0$ is the global optimum when $\text{ED}_A$ is used. Now we start with $A_0$ and increase or decrease its diagonal entries to find a matrix $A$ that works better than $A_0$ when $\text{DTW}_A$ is used. In order to mitigate the problem of a local optimum in hill-climbing search, we also conduct the search by multiple restarts with a random diagonal matrix $A$. We then compare the solutions for $A$ from both approaches and finally choose the one with the minimum objective function.

**(1) Initial State:** We perform experiments on two different initial states.

– Starting with $A_0$, the diagonal matrix from learning the transformed Euclidean distance $\text{ED}_A$.
– Starting with $A_{rand}$, a diagonal matrix with random diagonal entries between 0 and 1.

Note that the sum of diagonal entries of $A_0$ and $A_{rand}$ is normalized to 1, required by constraint (6). This can simply be done by dividing each entry by the sum of all diagonal entries.

**(2) Heuristic Function:** The heuristic function $h$ is used to evaluate the quality of an operation. We use as heuristic function the objective function (5) formulated in the optimization problem OP2, i.e.,



Initial State
$A_0(1,1)$ $A_0(2,2)$ $A_0(3,3)$
Calculate h(0)
Calculate h(1)
$A_0(1,1)+\alpha$ $A_0(2,2)-\alpha$ $A_0(3,3)$
$A_0(1,1)+\alpha$ $A_0(2,2)$ $A_0(3,3)-\alpha$
$A_0(1,1)-\alpha$ $A_0(2,2)+\alpha$ $A_0(3,3)$
......
$A_0(1,1)$ $A_0(2,2)-\alpha$ $A_0(3,3)+\alpha$
$A_0(1,1)+\alpha+\alpha$ $A_0(2,2)-\alpha$ $A_0(3,3)-\alpha$
......
$A_0(1,1)+\alpha$ $A_0(2,2)-\alpha$ $A_0(3,3)-\alpha+\alpha$
This state has minimum h(1) in level 1 and h(1) < h(0)

**Fig. 1.** Illustration of hill-climbing algorithm, $n = 3$. $\alpha$ is the step-size parameter

$$h = \frac{\sum\limits_{(s_i,s_j)\in ML} DTW_A^2(s_i,s_j)}{(\sum\limits_{(s_i,s_j)\in CL} DTW_A(s_i,s_j))^2} \tag{7}$$

**(3) Successor function:** Once the initial state has been determined, we want to improve that solution. A successor function generates a successor state of the current state. Figure 1 illustrates this progress. We define a step-size parameter $\alpha$, $0 \leq \alpha \leq 1$. Assume there are $n$ dimensions. In the current state, the weights of the corresponding dimensions are $A(1,1), A(2,2), \ldots, A(n,n)$. To generate the successor states, we increment the weight $A(i,i), 1 \leq i \leq n$ by $\alpha$ and decrement another weight $A(j,j), j \neq i, 1 \leq j \leq n$ by $\alpha$. In this case, the sum of the weights is always 1. We apply this step to each $(i,j)$ pair and pick the optimal successor state to survive and reproduce. According to our definition of the heuristic function, the optimal successor state is the state whose $h$ is minimal among the states at the same level. If it is better than the current state, let it reproduce. If not, return the current state.

**(4) Terminal test:** It defines the condition when to stop the search. We stop the search if no improvement can be made, and the current state is returned.

**The case of a full matrix *A*.** We now detail the learning approach for a full matrix $A$. The idea again is to use a hill-climbing algorithm. Similar to the learning of a diagonal matrix $A$, we start with the full matrix $A$ from learning of ED$_A$, denoted as $A_e$, and try to find a matrix $A$ that works better than $A_e$ with DTW$_A$.

First, we find the number of parameters that determine a real symmetric $n \times n$ full matrix $A$. With eigendecomposition, $A = ULU^T$. Since $A$ is real symmetric, we can always find an orthonormal matrix $U$ with $det(U) = 1$. Hence, both $U$ and $U^T$ are rotation matrices in an $n$-dimensional space. Similarly, with eigendecomposition, we obtain $A_e = U_e L_e U_e^T$, where $U_e$ is a rotation matrix. We can relate $U$ to $U_e$ as $U = BU_e$, where $B$ is also a rotation matrix. Hence, $A$

can be expressed as $A = BU_eLU_e^TB^T$. $A = A_e$ when $B = I_n, L = L_e$. Since $U_e$ is known, $A$ is determined by only two matrices $B$ and $L$. $L$ is parameterized by $n$ eigenvalues of $A$. As a rotation matrix in $n$-dimensional space, $B$ is determined by $\frac{n(n-1)}{2}$ angles. This is obvious in a 2-dimensional space where a simple rotation matrix is parameterized by one angle $\theta$ as $[\begin{smallmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{smallmatrix}]$. Generally, a rotation matrix in an $n$-dimensional space can be considered as a composition of the rotations occurring in each plane formed by any two coordinate axes, and there are $\frac{n(n-1)}{2}$ such planes. In an $n$-dimensional space, the rotation matrix that rotates the axis $X_i$ in the direction of $X_j$ by angle $\theta$ is $R_{i,j}(\theta) = \left\{ r_{ab} \mid \begin{array}{l} r_{i,i} = cos\theta \\ r_{j,j} = cos\theta \\ r_{i,j} = -sin\theta \\ r_{j,i} = sin\theta \\ r_{a,a} = 1, a \neq i, a \neq b \\ r_{a,b} = 0, \text{elsewhere} \end{array} \right\}$.

As a rotation matrix in $n$-dimensional space, $B$ can be obtained by building the product of all $R_{i,j}(\theta)$ [6]. As one can see, each matrix $R_{i,j}$ has one parameter $\theta$. Thus $B$ is parameterized by $\frac{n(n-1)}{2}$ angles. We therefore conclude that learning $A$ is equivalent to assigning values to $\frac{n(n-1)}{2}$ angles and $n$ eigenvalues.

We now determine the search space. According to constraint (6), $L_{i,i}$ (i.e., the eigenvalues of $A$) should satisfy $\sum_{i=1}^{n} L_{i,i} = 1$ and $L_{i,i} \geq 0, 1 \leq i \leq n$. Each angle is between 0 and $2\pi$. Let $\theta_1, \ldots, \theta_{\frac{n(n-1)}{2}}$ denote the angles that parameterize $B$. The components of the search algorithm are then as follows:

**(1) Initial State:** Similar to the learning of a diagonal $A$, there are two different initial states, corresponding to $A_e$ and a random matrix $A$, respectively.

(1) $\theta_i = 0$; $L = L_e$.

(2) Random $\theta_i$ and $L$ with random entries.

**(2) Heuristic Function:**

$$h = \frac{\sum_{(s_i,s_j)\in ML} DTW_A^2(s_i, s_j)}{(\sum_{(s_i,s_j)\in CL} DTW_A(s_i, s_j))^2} \tag{8}$$

**(3) Successor Function:** The successor function generates successor states by changing the values of the parameters in the current state. Suppose $\alpha_\theta$, $\alpha_L$ are the step-size parameters for $\theta_i$ and $L_i$, respectively. In the current state, suppose the angles are $\theta_1, \ldots, \theta_{\frac{n(n-1)}{2}}$ and the eigenvalues are $L_1, \ldots, L_n$. There are two available successor functions:

(a) Increasing $\theta_i$ by $\alpha_\theta$ if $\theta_i + \alpha_\theta \leq 2\pi$.
(b) Increasing $L_i$ by $\alpha_L$ and decrementing another eigenvalue $L_j$ by $\alpha_L$, to keep the sum of $L_i$ as 1.

(a) and (b) alternate in the process. The algorithm evaluates the successor states according to the heuristic function presented above and chooses the best state to continue the search.

**(4) Goal Test:** The search terminates if no successor state is better than the current state.

## 5 Experiments and Evaluation

In this section, we present the experiments and evaluations we conducted to demonstrate the flexibility and effectiveness of our distance learning approach. In Sec. 5.1, we discuss the adaptivity of our approach to user-specified constraints. In Sec. 5.2, we then show how our learning approach is used to improve clustering and classification performance for object trajectories. In our experiments, we use synthetic and real-world datasets that describe object trajectories.

For this, we compare the performance of the following six distance measures:

- (1) Euclidean distance ED and (2) Dynamic Time Warping DTW,
- Transformed Euclidean distance $\text{ED}_A$ with (3) diagonal and (4) full matrix $A$, and
- Transformed Dynamic Time Warping $\text{DTW}_A$ with (5) diagonal and (6) full matrix $A$.

We chose the measures ED and DTW, because Ding et al. have shown that EDR, LCSS, ERP etc. are not more accurate than the classic DTW in general [4]. Thus, if our learned distance measures beat ED and DTW, this suggests that they also beat the other distance measures. Therefore, we do not discuss measures other than ED and DTW in our experiments. Furthermore, we constrain the warping band of DTW and $\text{DTW}_A$ to up to 20% of the trajectory size as in [20]. We use $\alpha = \alpha_L = 0.1$, and $\alpha_\theta = \pi/8$ in this work.

### 5.1 Adaptivity to Constraints

In the following, we show that our learning approach produces distance measures that are adaptive to the constraints specified by users. Assume two users with different views on similarity of a given trajectory dataset and each user specifies a set of constraints reflecting his view. By first learning the distance measures from the constraints and then performing clustering using the learned distance measures, we show that our "single" learning approach can help both users find the results satisfying their views on trajectory similarity.

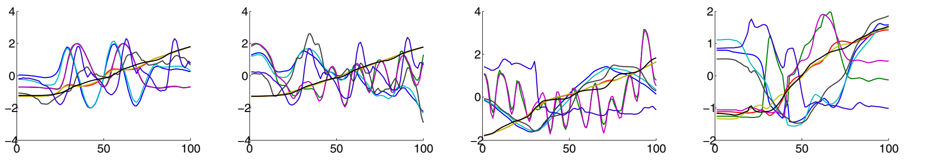
We use a real world trajectory dataset to verify the adaptivity of our approach to constraints. In addition to the clustering accuracy of the six different distance measures on this dataset, we also present the visual comparison between original and transformed trajectories to illustrate the flexibility of our approach.

For this experiment, we generated a trajectory dataset, called myMotion [26], that represents human motion. The myMotion dataset consists of human gait data captured by a VICON system. All the data were created using a female actor who had markers attached. The female actor performed 4 groups of movements, each group consisting of 9 examples. The camera recording rate was 60 frames per second. For each marker, its 3-d positions $x, y$ and $z$ in each frame were recorded. The 4 groups of motion data represent the following movements:

G1: Walking in a straight line and swinging one arm.
G2: Walking in a straight line and keeping arms stable.
G3: Walking in a Z-shape line and keeping arms stable.
G4: Walking in a Z-shape line and swinging one arm.

We use the data readings from 3 markers attached to the actor's body, one arm, and one leg. Each marker has 3 features, corresponding to the $x, y$, and $z$ positions. Thus, each example has 9 features. The myMotion dataset contains a total of 36 9-d time series data instances. All instances have the same length of 100, obtained through interpolation. Each dimension is Z-normalized, as a standard preprocessing step in motion matching.

Figure 2 shows four samples of the 9-d trajectories. Each of the four plots represents one sample from G1 to G4. In the figures, each dimension is plotted as a time series in a unique color. The $x$ axis represents the time $t$, and the $y$ axis corresponds to the coordinates.



(a) Sample from $G1$. (b) Sample from $G2$. (c) Sample from $G3$. (d) Sample from $G4$.

**Fig. 2.** Four **original** samples from the myMotion dataset

Suppose there are two users with different views of similarity and they both want to cluster the 36 trajectories into 2 clusters. For the first user, the "true clusters" are distinguished by the actor's arm movements. Therefore, he thinks that the data from $G1$ and $G4$ belong to one cluster and G2 and G3 belong to the other. On the other hand, the "true clusters" for the second user are distinguished by the actor's walking routine, i.e., Z-shape or straight line. He considers the trajectories from G1 and G2 being one cluster and the trajectories from G3 and G4 being the other cluster. Both users now specify their preferences in the form of $ML$ and $CL$ constraints, indicating their respective similarity views. In this experiment, we "simulate" the specification of user constraints by random sampling. That is, trajectories in a user's cluster are assumed to have the same label. Constraints $ML$ are random samples from all pairs of trajectories with the same label from the user's point of view, and constraints $CL$ include random samples of all pairs of trajectories having different labels.

For each user and his constraints $CL$ and $ML$, the learning approach generates the distance measures from his constraints. Then the trajectories are clustered according to the learned distance measures. In the clustering experiment, we use the group average hierarchical clustering algorithm to cluster the data. In the corresponding dendrogram, we look at the first (top) branch, which produces two subtrees, each representing one cluster. The cluster accuracy is given as

$$\text{Accuracy} = \frac{\text{number of correctly labeled data}}{\text{number of all data}}.$$

We compute the clustering accuracy based on ED, DTW and the four distance measures $\text{ED}_A$ and $\text{DTW}_A$ for a diagonal and full matrix $A$, respectively. Here, 4% of all pairs of trajectories having the same label (resp. different labels) are randomly sampled as $ML$ (resp. $CL$). We will talk more about the relation between the size of constraints and the clustering accuracy in Sec. 5.2. Table 1 lists the accuracy values for both users for the learned matrix $A$. As one can see, although the two users give two different sets of constraints, our approach achieves a very high clustering accuracy in both cases.

**Table 1.** Clustering accuracy for the myMotion dataset given two different sets of user constraints. The numbers in bold show the best clustering accuracy for each user. $\text{DTW}_A$ performs best for both users.

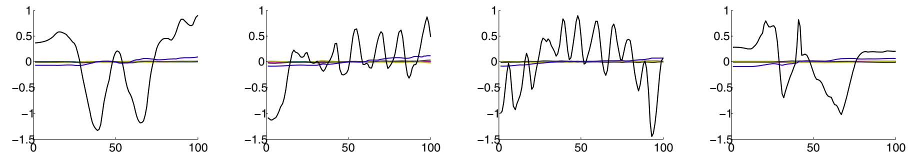
| | $ED_A$, diag. A | $ED_A$, full A | $DTW_A$, diag. A | $DTW_A$, full A | DTW | ED |
|---|---|---|---|---|---|---|
| User 1 | 83.3% | 94.4% | 97.22% | **100%** | 72.2% | 83.3% |
| User 2 | 61.1% | 55.56% | **97.22%** | 94.4% | 72.2% | 61.1% |

Recall that learning the matrix $A$ is equivalent to finding a transformation $W$ ($W = A^{1/2}$) of the trajectories so that the pairs in $ML$ are more similar to each other and the pairs in $CL$ are different from each other. To visually illustrate this aspect, we plot the transformed trajectories learned from the first and second user constraints, respectively. The transformation corresponds to the distance measure leading to the best clustering accuracy for each case. In other words, the transformation in Fig. 3 is based on the full matrix $A$ with $\text{DTW}_A$ learned from the first user's constraints, and the transformation in Fig. 4 corresponds to the diagonal matrix $A$ with $\text{DTW}_A$ learned from the second user's constraints.

As one can see in Fig. 3, after the transformation, the trajectory samples from G1 (Fig. 3(a)) and G4 (Fig. 3(d)) are very similar to each other. Also the samples from G2 (Fig. 3(b)) and G3 (Fig. 3(c)) look similar now. On the other hand, in Fig. 4, the samples from G1 (Fig. 4(a)) and G2 (Fig. 4(b)) are similar and so are the samples from G3 (Fig. 4(c)) and G4 (Fig. 4(d)).

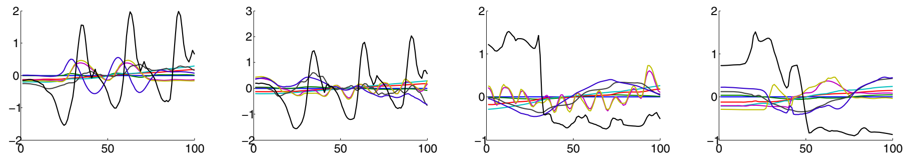
## 5.2 Improvement of Accuracy for Clustering and Classification

The other important application of our approach is to improve the accuracy in the context of clustering and classification. To demonstrate this feature, we performed experiments on datasets from different application domains.



(a) Sample from $G1$. (b) Sample from $G2$. (c) Sample from $G3$. (d) Sample from $G4$.

**Fig. 3.** Four **transformed** samples. The transformation corresponds to a full matrix $A$ when $\mathrm{DTW}_A$ is learned from the first user's constraints. Samples from G1 and G4 show similarity, and the G2 sample is very similar to the G3 sample.



(a) Sample from $G1$. (b) Sample from $G2$. (c) Sample from $G3$. (d) Sample from $G4$.

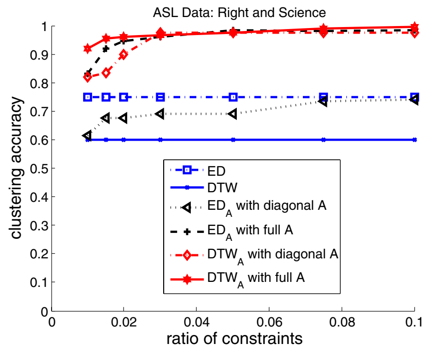
**Fig. 4.** Four **transformed** samples. The transformation corresponding to a diagonal matrix $A$ for $\mathrm{DTW}_A$ is learned from the second user's constraints. Samples from G1 and G2 show similarity, and the G3 sample is very similar to the G4 sample.

**Application to Clustering.** First, we show that our method can be applied to improve the accuracy of clustering. Assume a trajectory dataset $S = \{s_i\}_{i=1}^{u}$ and constraints $ML$ and $CL$ specified by a user. As mentioned before, $(s_i, s_j) \in ML$ means that the user considers $s_i$ and $s_j$ to belong to the same cluster, and for $(s_i, s_j) \in CL$, $s_i$ and $s_j$ belong to different clusters. We applied our approach to two labeled multi-dimensional time series datasets, ASL and Trace.

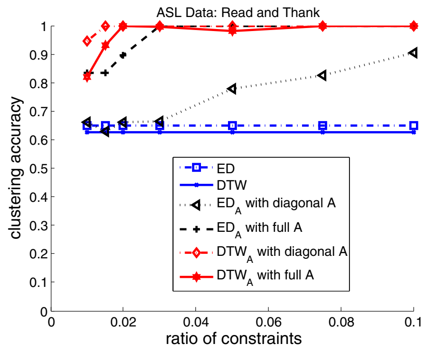
The Australian Sign Language (ASL) dataset [13] consists of the hand trajectories of a native ASL speaker when he expressed signs. We use the cleaned dataset from the UCR data archive [14]. The dataset has 10 classes, and each class has 20 examples. The 20 examples in each class represent the same word in ASL, each example having 8 features. All trajectories are interpolated to the length of 30. In this clustering experiment, we use two pair of classes, which represent the signs "read" and "thank", "right" and "science", respectively.

The Transient Classification Benchmark (Trace) dataset [14] is synthetic and has 16 classes. We use the class pairs (2 and 3), and (6 and 7) in the clustering experiments. We use 20 instances from each class, where each instance has 4 features. All data are interpolated to the same length of 50.

For each pair of classes, we combine the data into one set and perform the group-average hierarchical clustering algorithm to distinguish them. The computation of the clustering accuracy is the same as the one used for myMotion dataset in Sec. 5.1. With each dataset, the $ML$ and $CL$ constraints are generated as follows. $ML$ is generated by selecting a random subset of all pairs of data having the same class label. Analogously, $CL$ includes a random subset of all pairs of data with different class labels.



(a) Words 'Right' and 'Science' from the ASL dataset



(b) Words 'Read' and 'Thank' from the ASL dataset

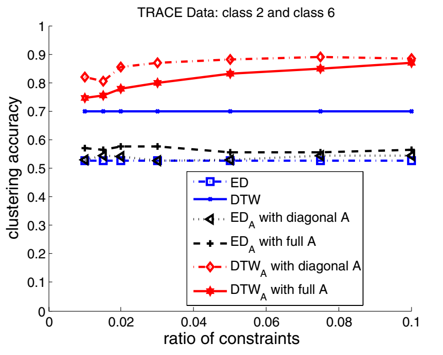
**Fig. 5.** Clustering accuracy vs size of constraints; the $x$ axis is the fraction of all pairs of data sharing the same (different) class label(s) that are sampled to be included in $ML$ ($CL$); the $y$ axis shows the clustering accuracy

In order to discover the relationship between clustering accuracy and size (i.e., number) of constraints, we generated sets of constraints of different sizes and determined the clustering accuracy for each set. We repeated the experiment five times, because of the randomness of the selection of constraints. The average accuracy of the five trials is shown in Fig. 5, which shows the plot of clustering accuracy vs. size of constraints for the ASL dataset. The results for the Trace dataset are shown in Fig. 6.

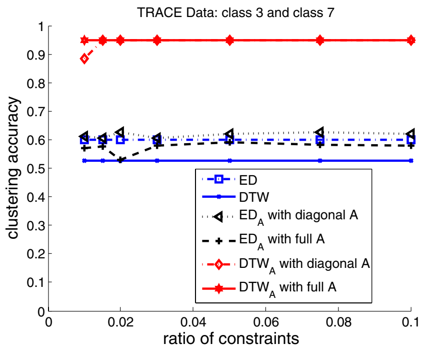
As one can see in Fig. 5 and Fig. 6, the clustering accuracy is obviously affected by the size of the constraints. We tried constraint sizes from 1% to 10% of all pairs of data with the same label. Clearly the accuracy of ED and DTW does not change with the size of the constraints as they do not consider constraints. Both ED$_A$ with diagonal and full matrix $A$ and DTW$_A$ with diagonal and full matrix $A$ have a better performance when the number of constraints increases, except in Fig. 6(b), where DTW$_A$ with full matrix $A$ achieves high accuracy even when the constraint size is very small. In Fig. 5, when the number of constraints is small, DTW$_A$ performs better than ED$_A$. If we specify a larger size of constraints, ED$_A$ achieves a performance comparable to the one of DTW$_A$. However, for the Trace dataset, whose results are shown in Fig. 6, DTW$_A$ always performs better than ED$_A$. The reason for this is that there is more distortion in the time axis in the Trace dataset than in the ASL dataset.

**Application to Classification.** The learning approach presented in this paper can also be used to improve the accuracy of classification. In this experiment, we use three labeled multi-dimensional time series datasets.

50*CommonWords data.* The 50CommonWords dataset from the UCR data repository [14] contains 50 distinct words, each of which has various handwriting instances. Each instance is represented by 4 features describing the handwriting of a word. The instances of the same word are considered data having the same



(a) Classes 2 and 6 in the Trace dataset



(b) Classes 3 and 7 in the Trace dataset

**Fig. 6.** Clustering accuracy vs size of constraints; the $x$ axis is the fraction of all pairs of data having the same (different) class label(s) that are sampled to be included in $ML$ ($CL$); the $y$ axis shows the clustering accuracy

label. For simplicity, we picked the instances of the commonly used words "of", "be", and "at". "of" has 54 instances, "be" has 38 instances, and "at" has 22 instances. All instances were interpolated to the length of 50.

*ASL dataset.* Here we used the classes "girl", "come" and "name", which are different from the data in the clustering experiment.

*Trace dataset.* We used the classes 9, 11, and 13 here to use data different from the clustering experiment.

We evaluate the application of our distance measure learning approach to classification by using an objective evaluation framework proposed by Ding et al. [4]. The general idea is to use a cross validation method and a 1-nearest neighbor (1NN) classifier for the labeled data to evaluate the accuracy of the learned distance measure. Assume a labeled dataset partitioned into a training dataset and a test dataset. For each data item in the test dataset, we predict its label to be the same as the label of its nearest neighbor in the training dataset. If the predicted label is the same as the actual label, it is considered a hit, otherwise it is considered a miss.

In order to conduct the $k$ cross validation, the labeled dataset is randomly partitioned into $k$ sets. The $k$ cross validation has $k$ runs. In each run, one of the $k$ sets is chosen as the training dataset, and the other $k-1$ sets are used as the testing dataset. The $ML$ and $CL$ constraints are generated from the training dataset in the same way as in the clustering experiments described above. For the generated constraints, the distance learning algorithms are invoked to generate the distance measures (1) $\text{ED}_A$ with a diagonal matrix $A$ and a full matrix $A$, and (2) $\text{DTW}_A$ parameterized with a diagonal matrix $A$ and a full matrix $A$. For each of the resulting four distance measures, we conduct the 1NN classification to evaluate its accuracy. We also apply the 1NN classification to the pre-defined distance measures ED and DTW for comparison. We used the

**Table 2.** The classification accuracy of six distance measures for the three datasets. The numbers in bold indicate the best accuracy for each dataset. In general, for all datasets, the best accuracy is obtained by using the learned distance measures.

| | ED | DTW | ED$_A$, diag. A | ED$_A$, full A | DTW$_A$, diag. A | DTW$_A$, full A |
|---|---|---|---|---|---|---|
| ASL | 78.75% | 75% | **95.42%** | 93.75% | 91.67% | 90.83% |
| Trace | 39.58% | 52.92% | 41.67% | 40.27% | **60.42%** | 57.50% |
| 50Com. | 90.94% | 95% | 91.25% | 90.94% | **95.94%** | 95.31% |

LB-Keogh lower bounding [15] to speed up the computation for DTW and DTW$_A$ and the following formula to compute the accuracy for each run:

$$\text{Accuracy} = \frac{\text{number of correctly labeled testing data}}{\text{number of all testing data}}.$$

Because there are $k$ runs, we compute the average of the accuracy of $k$ runs, as recommended by Ding et al. in [4]. The accuracy measures how well the predicted labels match the actual labels of the testing data. In this experiment, we use $k = 5$, which is within the range recommended by [4] to minimize the bias and variation. Table 2 lists the classification accuracy for all the datasets. As one can see, for all datasets, the best accuracy is obtained by using the learned distance measures. DTW$_A$ performs best on the Trace and 50CommonWords datasets and ED$_A$ performs best for the ASL dataset.

Note that ED$_A$ performs better than (resp. equal to) DTW$_A$ on the ASL dataset in classification (resp. clustering). This is consistent with the fact that ED performs better than DTW on the ASL dataset in all experiments, indicating that there is not much distortion in the time axis in this dataset. In a practical use, if one can obtain some more information about the time distortion in the dataset, one then can decide which distance measure performs better.

## 6 Conclusions and Ongoing Work

Distance measures play an important role in many data mining and analysis tasks for multi-dimensional object trajectories. Instead of relying on some of the existing, hard-coded distance measures and to support different user views on what trajectories are (dis)similar in a particular domain, in this paper, we presented a comprehensive approach for learning distance measures from user constraints. A key idea is to pose the proposed learning approach as an optimization problem that effectively utilizes well-known techniques. Our evaluations demonstrate that the learned transformed Euclidean and transformed DTW not only provide a high degree of adaptivity to user constraints but also achieve a high degree of accuracy in clustering and classification tasks for object trajectories. In general, the proposed techniques provide much more flexibility to support different user views than existing approaches.

We are currently studying the performance of the learning approach using additional datasets from other domains, in order to better evaluate the choice between transformed Euclidean and transformed DTW distance. Another interesting aspect, along the line of [20], is to derive an index structure for indexing the trajectories once a distance measure has been learned.

**Acknowledgments.** This work is in part supported by the National Science Foundation under Award No. ATM-0619139.

## References

1. Berndt, D.J., Clifford, J.: Using Dynamic Time Warping to Find Patterns in Time Series. In: AAAI Workshop on Knowledge Discovery in Databases (1994)
2. Chen, L., Ng, R.T.: On the marriage of Lp-norms and edit distance. In: VLDB, pp. 792–803 (2004)
3. Chen, L., Özsu, M.T., Oria, V.: Robust and fast similarity search for moving object trajectories. In: ACM SIGMOD 2005, pp. 491–502 (2005)
4. Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E.J.: Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures. In: VLDB 2008, pp. 1542–1552 (2008)
5. Domeniconi, C., Gunopulos, D.: Adaptive Nearest Neighbor Classification using Support Vector Machines. In: Advances in Neural Information Processing Sytems, vol. 13, pp. 665–672. MIT Press, Cambridge (2002)
6. Duffin, K.L., Barrett, W.A.: Spiders: A New User Interface for Rotation and Visualization of N-dimensional Point Sets. In: IEEE Visualization 1994, pp. 205–211 (1994)
7. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast subsequence matching in time-series databases. In: ACM SIGMOD 1994, pp. 419–429 (1994)
8. Frentzos, E., Gratsias, K., Theodoridis, Y.: Index-based Most Similar Trajectory Search. In: ICDE, pp. 816–825 (2007)
9. Grumbach, S., Rigaux, P., Segoufin, L.: Manipulating Interpolated Data is Easier than you Thought. In: VLDB 2000, pp. 156–165 (2000)
10. Han, J., Kamber, M.: Data Mining: Concepts and Techniques, 2nd edn. Morgan Kaufmann, San Francisco (2006)
11. Lee, J.G., Han, J., Whang, K.Y.: Trajectory clustering: a partition-and-group framework. In: SIGMOD 2007, pp. 593–604 (2007)
12. Lee, S.L., Chun, S.J., Kim, D.H., Lee, J.H., Chung, C.W.: Similarity Search for Multidimensional Data Sequences. In: ICDE 2000, pp. 599–608 (2000)
13. Kadous, M.W.: Temporal Classification: Extending the Classification Paradigm to Multivariate Time Series. Ph.D. Thesis, School of Computer Science and Engineering, University of New South Wales (2002)
14. Keogh, E.J.: The UCR Time Series Data Mining Archive, University of California at Riverside, http://www.cs.ucr.edu/~eamonn/TSDMA
15. Keogh, E.J.: Exact Indexing of Dynamic Time Warping. In: VLDB 2002, pp. 406–417 (2002)
16. Korn, F., Jagadish, H.V., Faloutsos, C.: Efficiently supporting ad hoc queries in large datasets of time sequences. In: ACM SIGMOD 1997, pp. 289–300 (1997)
17. Mokbel, M.F., Aref, W.G.: Location-aware Query Processing and Optimization: A Tutorial. In: Presented at the 8th Int. Conf. on Mobile Data Management (2007)
18. Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. IEEE Transactions on Acoustics, Speech and Signal Processing 26(1), 43–49 (1978)

19. Tsotras, V.J.: Recent Advances on Querying and Managing Trajectories. In: Tutorial at the 10th International Symposium on Spatial and Temporal Databases (2007)
20. Vlachos, M., Hadjieleftheriou, M., Gunopulos, D., Keogh, E.J.: Indexing multi-dimensional time-series with support for multiple distance measures. In: KDD 2003, pp. 216–225 (2003)
21. Vlachos, M., Gunopulos, D., Das, G.: Rotation invariant distance measures for trajectories. In: KDD 2004, pp. 707–712 (2004)
22. Fu, A.W.-C., Keogh, E.J., Lau, L.Y.H., Ratanamahatana, C.: Scaling and time warping in time series querying. In: VLDB 2005, pp. 649–660 (2005)
23. Xing, E., Ng, A., Jordan, M., Russell, S.: Distance metric learning, with application to clustering with side-information. In: Advances in Neural Information Processing Systems, vol. 15, pp. 505–512 (2003)
24. Yanagisawa, Y., Akahani, J., Satoh, T.: Shape-Based Similarity Query for Trajectory of Mobile Objects. In: 4th Int. Conf. on Mobile Data Management, pp. 63–77 (2003)
25. Yang, L.: Distance Metric Learning: A Comprehensive Survey. Dept. of Comp. Science and Eng. Michigan State University (2006)
26. `http://wwwcsif.cs.ucdavis.edu/~yuwei/learning.html`

# Author Index

Agrawal, Gagan 74, 218
Ailamaki, Anastasia 148
Akal, Fuat 185
Anand, Manish Kumar 237
Anderson, Erik W. 292
Archner, Oliver 255
Assent, Ira 497
Aßfalg, Johannes 435

Barreto, Roselyne 273
Baumann, Peter 56
Bethel, E. Wes 110
Bettini, Claudio 562
Bouros, Panagiotis 398
Bowers, Shawn 237
Brenninkmeijer, Christian Y.A. 339
Burke, John 37
Burns, Randal 148

Cary, Ariel 302
Chatzopoulou, Gloria 3
Chaudhary, Amitabh 148
Chen, Guanhua 580
Chiu, David 218
Cohen-Boulakia, Sarah 283
Critchlow, Terence 37
Curé, Olivier 255

Dalamagas, Theodore 361, 398
Dash, Debabrata 148
Dayal, Umeshwar 479

Eirinaki, Magdalini 3

Fernandes, Alvaro A.A. 339
Fomkin, Ruslan 320
Freire, Juliana 292

Galpin, Ixent 339
Gertz, Michael 627
Golfarelli, Matteo 83
Gosink, Luke J. 110
Grant, Kevin 553
Green-Fishback, Harrison 19
Gupta, Chetan 479
Gupta, Indranil 167
Gutell, Robin R. 200

Han, Jiawei 167
Hengartner, Nick 37
Hogan, Emilie 37
Howe, Bill 19
Hristidis, Vagelis 302

Ioup, Elias 553

Jabeen, Farhana 339
Jablonski, Stefan 255
Jin, Xin 167
Joho, Dieter 185
Joslyn, Cliff 37
Joy, Kenneth I. 110

Katzdobler, Alexander 444
Keahey, Kate 478
Kim, Sangkyum 167
Klasky, Scott 273
Koop, David 292
Kosar, Tevfik 191
Kröger, Peer 435, 444
Kriegel, Hans-Peter 435, 444

Li, Yuan 461
Lin, Jessica 461
Liu, Eric Yi 517
Lu, Hua 608
Ludäscher, Bertram 217, 237

Ma, Xiuli 580
Maier, David 19
Malik, Rahul 167
Malik, Tanu 148
Masini, Kevin 283
Mattoso, Marta 264
McKenney, Mark 598
McPhillips, Timothy 237
Mehta, Abhay 479
Mouallem, Pierre 273
Murta, Leonardo 264
Müller, Emmanuel 497

Nahrstedt, Klara 167
Normand, Eric 553

Ogasawara, Eduardo 264
Otoo, Ekow 92
Owens, John D. 110
Ozer, Stuart 200

Paton, Norman W. 339
Paulino, Carlos 264
Podhorszki, Norbert 273
Pola, Ives R.V. 417
Polyzotis, Neoklis 3

Ramachandran, Chandrasekar 167
Rehman, M. Abdul 255
Renz, Matthias 435, 444
Riboni, Daniele 562
Risch, Tore 320
Rishe, Naphtali 302
Rizzi, Stefano 83
Rotem, Doron 92
Rundensteiner, Elke 380

Sacharidis, Dimitris 398
Sample, John 553
Santos, Emanuele 292
Sarkar, Abhishek 517
Schlapbach, Ralph 185
Seidl, Thomas 497
Sellis, Timos 361, 398
Shoshani, Arie 1
Shuai, Meng 580
Siddiqui, Zaigham Faraz 535
Silva, Cláudio 292
Sinha, Rishi Rakesh 130
Skiadopoulos, Spiros 398
Souldatos, Stefanos 361
Spiliopoulou, Myra 535
Sun, Zhengguo 302

Tang, Shiwei 580
Theodoratos, Dimitri 361
Traina, Agma J.M. 417
Traina Jr., Caetano 417
Tsao, Shih-Chiang 92
Türker, Can 185

Vo, Huy T. 292
Volz, Bernhard 255
Vouk, Mladen 273

Wang, Fan 74
Wang, Song 479
Wang, Wei 517
Wang, Xiaodan 148
Wang, Xinqi 191
Wei, Mingrui 380
Werner, Cláudia 264
Winslett, Marianne 130
Wu, Kesheng 110, 130
Wu, Xiaoying 361

Xu, Weijia 200

Yang, Dongqing 580
Yiu, Man Lung 608
Yu, Wei 627

Zhang, Qi 517
Züfle, Andreas 444