

Alvaro A.A. Fernandes
Alasdair J.G. Gray
Khalid Belhajjame (Eds.)

LNCS 7051

Advances in Databases

28th British National Conference on Databases, BNCOD 28
Manchester, UK, July 2011
Revised Selected Papers

 Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Alfred Kobsa

University of California, Irvine, CA, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

TU Dortmund University, Germany

Madhu Sudan

Microsoft Research, Cambridge, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max Planck Institute for Informatics, Saarbruecken, Germany

Alvaro A.A. Fernandes
Alasdair J.G. Gray
Khalid Belhajjame (Eds.)

Advances in Databases

28th British National Conference on Databases, BNCOD 28
Manchester, UK, July 12-14, 2011
Revised Selected Papers



Springer

Volume Editors

Alvaro A.A. Fernandes
Alasdair J.G. Gray
Khalid Belhajjame
University of Manchester
Manchester M13 9PL, UK
E-mail: a.fernandes@manchester.ac.uk
{a.gray, khalid.belhajjame}@cs.man.ac.uk

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-24576-3 e-ISBN 978-3-642-24577-0
DOI 10.1007/978-3-642-24577-0
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011939304

CR Subject Classification (1998): H.4, H.3, H.2, H.2.8, I.2.4, I.2.6

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

© Springer-Verlag Berlin Heidelberg 2011

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The 28th British National Conference on Databases (BNCOD 2011) was held during July 2011 in the School of Computer Science at the University of Manchester. This volume contains the proceedings of the conference and consists of revised versions of presented contributions.

Following a recent tradition in the long history of BNCOD, the conference was given a theme, *Linked Data*, a topic that has gained attention as a concrete aspect of the Semantic Web. The linked data approach focusses on moving from a Web of documents to a Web of richly annotated data. This vision is being actively pursued by many groups from both the Web and the information management research communities. Prominent among these is Christian Bizer's group at the Free University, Berlin. We were extremely grateful that Prof. Bizer attended the conference and gave the opening keynote speech. He spoke with great passion of the vast landscapes opened up by the principled combination of novel techniques in database research, such as the dataspace approach to semantic integration, and this brave new world of linked data, founded on standards such as RDF and SPARQL.

Stressing the crucial role of semantics in information management, particularly in peer-to-peer contexts, the closing keynote speech was given by Karl Aberer, from the École Polytechnique Fédérale de Lausanne, to whom we were extremely grateful for a deep and entertaining talk. Prof. Aberer's distinguished work is now reaching out to exciting new developments whose significance the BNCOD audience greatly appreciated.

BNCOD 2011 hosted one tutorial on *Mining Image Databases by Content* given by Gerald Schaefer, from Loughborough University, UK. Dr. Schaefer enlightened the audience with a sweeping pass through the latest advances in the area of image retrieval by content, focussing in particular on the role of color and texture.

Once again, the 9th International Workshop in Teaching, Learning, and Assessment of Databases (TLAD) was co-located with BNCOD. Nine papers were presented, one of which, voted the best by the workshop delegates, was selected for publication in these proceedings.

The PhD Forum is a regular event at BNCOD and this year was a particular exciting one, with six students presenting the current state of their work to an expert panel and an audience of their peers who were able to offer helpful advice and guidance.

The main body of the conference consisted of four full-length research paper sessions, one short-paper session, and a poster/demo session. These ranged over many of the hottest topics in information management at the time of writing: XML compression, XML updates, column-oriented stores, provenance,

warehousing, streamed data, data mashups, dataspace, sensor network query processing and pattern-oriented search.

There were 44 submissions to BNCOD 2011 from 18 countries. Each submission was blind reviewed by at least three Program Committee members. The Program Committee decided to accept 18 contributions (13 full, 2 short papers, 2 demo papers and 1 poster paper). The short papers, the demos and posters were submitted as such, i.e., with that intention from the start. The text of these contributions plus the best paper from TLAD are printed in full in this volume. The two keynote speeches and the tutorial are recorded here in the form of abstracts.

As Program Committee Chair, I would like to thank, first and foremost, my co-editors, Alasdair J.G. Gray and Khalid Belhajjame for their sterling work in several facets of BNCOD 2011. I would also like to thank the keynote and tutorial speakers for their contributions, the authors for choosing BNCOD as the venue for reporting their groundbreaking work, the Program Committee members for their thorough reviewing, and the BNCOD Steering Committee members for the opportunity given to us in Manchester to, once more, organize an edition of BNCOD.

Thanks are also due to our colleagues in Manchester who are behind the Easy-Chair conference management system, whose usefulness is now well recognized by so many in the world of science.

Finally, I would like to thank the delegates to the conference, whose involvement and participation helped make the event a memorable and rewarding one for all involved.

July 2011

Alvaro A.A. Fernandes

Organization

Program Committee

David Bell	Queen's University Belfast, UK
Sharma Chakravarthy	The University of Texas at Arlington, USA
Richard Cooper	University of Glasgow, UK
Alfredo Cuzzocrea	University of Calabria, Italy
Barry Eaglestone	University of Sheffield, UK
Suzanne Embury	University of Manchester, UK
Leonidas Fegaras	University of Texas at Arlington, USA
Ixent Galpin	University of Manchester, UK
Mary Garvey	University of Wolverhampton, UK
Georg Gottlob	Oxford University, UK
Anastasios Gounaris	Aristotle University of Thessaloniki, Greece
Alex Gray	Cardiff University, UK
Giovanna Guerrini	DISI- University of Genoa, Italy
Jun Hong	Queen's University Belfast, UK
Anne James	Coventry University, UK
Keith Jeffery	STFC, UK
Graham Kemp	Chalmers University of Technology, Sweden
Jessie Kennedy	Napier University, UK
Lachlan Mackinnon	University of Greenwich, UK
Nigel Martin	Birkbeck, University of London, UK
Marta Mattoso	COPPE- Federal University of Rio de Janeiro, Brazil
Peter Mcbrien	Imperial College London, UK
Marco Mesiti	DICO - University of Milan, Italy
Ken Moody	University of Cambridge, UK
David Nelson	University of Sunderland, UK
Werner Nutt	Free University of Bozen-Bolzano, Italy
Norman Paton	University of Manchester, UK
Alexandra Poulouvassilis	Birkbeck College, University of London, UK
Mark Roantree	Dublin City University, Ireland
Sandra Sampaio	The University of Manchester, UK
Alan Sexton	Birmingham University, UK
Jianhua Shao	Cardiff University, UK
Stratis Viglas	University of Edinburgh, UK
John N. Wilson	University of Strathclyde, UK

Additional Reviewers

Cappellari, Paolo
Cavalieri, Federico
Oliveira, Daniel
Peng, Taoxin
Razniewski, Simon
Valtolina, Stefano

Table of Contents

Evolving the Web into a Global Data Space (Abstract)	1
<i>Christian Bizer</i>	
Data Integration in a Networked World (Abstract)	2
<i>Karl Aberer</i>	
Reliable Provenance Information for Multimedia Data Using Invertible Fragile Watermarks	3
<i>Martin Schuler, Sandro Schulze, Ronny Merkel, Gunter Saake, and Jana Dittmann</i>	
ECOS: Evolutionary Column-Oriented Storage	18
<i>Syed Saif ur Rahman, Eike Schallehn, and Gunter Saake</i>	
X-HYBRIDJOIN for Near-Real-Time Data Warehousing	33
<i>Muhammad Asif Naeem, Gillian Dobbie, and Gerald Weber</i>	
Achieving High Freshness and Optimal Throughput in CPU-Limited Execution of Multi-join Continuous Queries	48
<i>Abhishek Mukherji, Elke A. Rundensteiner, and Matthew O. Ward</i>	
Mining Image Databases by Content	66
<i>Gerald Schaefer</i>	
Searching for Complex Patterns over Large Stored Information Repositories	68
<i>Nikhil Deshpande, Sharma Chakravarthy, and Raman Adaikkalavan</i>	
Using the Euclidean Distance for Retrieval Evaluation	83
<i>Shengli Wu, Yaxin Bi, and Xiaoqin Zeng</i>	
Expanding Sensor Networks to Automate Knowledge Acquisition	97
<i>Kenneth Conroy, Gregory C. May, Mark Roantree, and Giles Warrington</i>	
Utilising the MISM Model Independent Schema Management Platform for Query Evaluation	108
<i>Cornelia Hedeler and Norman W. Paton</i>	
Mining Sequential Patterns from Probabilistic Databases by Pattern-Growth	118
<i>Muhammad Muzammal</i>	

Smile: Enabling Easy and Fast Development of Domain-Specific Scheduling Protocols	128
<i>Christian Tilgner, Boris Glavic, Michael Böhlen, and Carl-Christian Kanne</i>	
On Integrating Data Services Using Data Mashups	132
<i>Muhammad Intizar Ali, Reinhard Pichler, Hong-Linh Truong, and Schahram Dustdar</i>	
Executing In-network Queries Using SNEE	136
<i>Ixent Galpin, Robert Taylor, Alasdair J.G. Gray, Christian Y.A. Brenninkmeijer, Alvaro A.A. Fernandes, and Norman W. Paton</i>	
Extracting Data Records from Query Result Pages Based on Visual Features	140
<i>Daiyue Weng, Jun Hong, and David A. Bell</i>	
Updates on Grammar-Compressed XML Data	154
<i>Alexander Bätz, Stefan Böttcher, and Rita Hartel</i>	
Reverting the Effects of XQuery Update Expressions	167
<i>Federico Cavaliere, Giovanna Guerrini, and Marco Mesiti</i>	
TraCX: Transformation of Compressed XML	182
<i>Stefan Böttcher, Rita Hartel, and Sebastian Stey</i>	
Satisfiability of Simple XPath Fragments under Fixed DTDs	194
<i>Nobutaka Suzuki</i>	
Computing Compressed XML Data from Relational Databases	209
<i>Stefan Böttcher, Dennis Bokermann, and Rita Hartel</i>	
Data Mining Project: A Critical Element in Teaching, Learning and Assessment of a Data Mining Module	221
<i>Hongbo Du</i>	
Author Index	237

Evolving the Web into a Global Data Space

Christian Bizer

Freie Universität Berlin

Germany

`chris@bizer.de`

Abstract. Linked Data technologies provide for setting links between records in distinct databases and thus to connect these databases into a global data space. Over the last years, Linked Data technologies have been adopted by an increasing number of data providers, including the US and UK governments as well as mayor players in the media and pharmaceutical industry, leading to the creation of a global Web of Data. In his talk, Prof. Christian Bizer will give an overview of the Web of Data as well as the architecture of Linked Data applications that work on top of this data space. Afterwards, he will discuss how the openness and self-descriptiveness of Linked Data provide for splitting data integration costs between data publishers, data consumers and third parties and thus might enable global-scale data integration in an evolutionary, pay-as-you-go fashion. He will close with an overview of the research challenges that the Linked Data community currently faces.

Data Integration in a Networked World

Karl Aberer

School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
`karl.aberer@epfl.ch`

Abstract. Traditionally, data integration techniques involve central components, e.g., global schemas or ontologies, to overcome semantic heterogeneity for enabling transparent access to heterogeneous data sources. Today, however, with the explosion of machine processable formats in the Data Web, one cannot rely on global, centralized schemas anymore, as knowledge creation and consumption are getting more and more dynamic and decentralized.

Peer-to-peer data integration systems are a good example of this new breed of systems eliminating central semantic components and replacing them through decentralized processes of local schema alignment and query processing. As a result semantic interoperability becomes an emergent property of the system.

In this talk we will first survey recent developments in peer-to-peer data integration, illustrating which novel challenges and opportunities these systems introduce. We then illustrate of how semantic integration can be modeled as a self-organizing agreement process and present a probabilistic technique for distributed reasoning about semantic relationships. We show how to extend these methods from the problem of schema mapping to entity matching and how to consider trustworthiness of autonomous participants in this process. Finally we present some recent work on applying these principles in business applications.

Reliable Provenance Information for Multimedia Data Using Invertible Fragile Watermarks*

Martin Schäler, Sandro Schulze, Ronny Merkel, Gunter Saake, and Jana Dittmann

School of Computer Science
University of Magdeburg, Germany
{schaeler, sansschul, ronny.merkel, saake,
jana.dittmann}@iti.cs.uni-magdeburg.de

Abstract. Today, more and more data is available in digital form, ranging from normal text to multimedia data such as image or video data. Since some data is of high sensitivity or undergoes legal restrictions, it is important to obtain more reliable information about the data origin and its transformations, known as data provenance. Unfortunately, current approaches for data provenance neither support multimedia data nor provide mechanisms to ensure reliability of the provenance information. In this paper, we present an approach based on existing watermarking schemes evaluated by a database system. Hence, this approach ensures the reliability of multi media data (e.g., fingerprint data) and its corresponding provenance information. Furthermore, we show how this approach can be applied within a specific database, used for fingerprint verification.

Keywords: Data Provenance, Multi Media Data, Invertible Watermarking.

1 Introduction

In the last decade, *data provenance* (a.k.a. *data lineage*), that is tracking the source and transformation of data in a predefined way [18], gained much attention in data-intensive systems. The reasons are twofold: On the one hand, more and more data is available from decentralised data sources such as the Internet or cloud computing. On the other hand, this data is liable to legal restriction or is of high sensitivity at all (e.g., biometric data). Hence, there is a growing interest on information about the origin of data and how it was created. According to the notion introduced by Glavic et al., we refer to the first as *source provenance* and to the latter as *transformation provenance* of data [16].

There are different application domains that have different requirements to the type of data provenance and the querying and manipulation facilities of provenance information. Prominent application domains that are covered by data provenance research are curated databases, data warehouses, geo information systems, workflow management systems or copyright identification [3,11]. Depending on the domain, data provenance therefore provides valuable information about the integrity or derivation/transformation process of the data.

However, current data provenance approaches have some limitations. Firstly, they are mostly applicable to structured or semi-structured data such as in relational databases.

* This work has been funded by the German Federal Ministry of Education and Science (BMBF) through the Research Programme under Contract No. FKZ:13N10817 and FKZ:13N10818.

Secondly, most approaches do not care about the reliability of provenance information. For instance, information on the original source(s) or transformation of a data item¹ may be tampered, e.g., through manipulation by an unauthorised instance. As a result, the authenticity (i.e., the data is what it claims to be) and integrity (i.e., no unauthorised changes) of the provenance information as well as of the corresponding data is not guaranteed anymore.

In this paper, we address both of these limitations. Therefore, we introduce a motivating example by considering fingerprint data. Then, we present an approach that uses watermarking schemes, such as as the scheme of Merkel et al. [22], to provide reliable provenance information for multimedia data. Furthermore, we show how to extend an existing database system to analyze the watermark. In particular, we make the following contributions:

- A new use case that goes beyond the traditional application of data provenance.
- We introduce an approach, that uses well-known invertible digital watermarking schemes in the context of databases, to gather and store provenance information for multimedia data. This approach ensures that the data as well as the corresponding provenance information holds integrity and authenticity.
- We initiate discussion on the usage of watermarking for reliable data provenance.

2 Background

Subsequently, we give background information regarding data provenance, watermarking in general and invertible watermarking techniques.

2.1 Data Provenance

In data provenance, we distinguish between *provenance model* and *provenance management system*. The first describes the conceptual model for provenance while the latter describes a system for the management of provenance information. The provenance model is fundamental to all data provenance systems, because it defines all aspects of how provenance information is recorded and processed. There are different provenance management systems that are mostly tailored to their application domain. For a more detailed overview, we refer to the work of Simmhan et al. [23]. In the following, we introduce some aspects of data provenance that are important for the work, proposed in this paper. For a detailed overview of data provenance aspects, we refer to several surveys [16,27,17].

Provenance information can be recorded for data items with different granularity. In this context, a data item is any structural unit of data, e.g., a tuple or a relation regarding a relational database. Furthermore, there are two general views on data provenance. The first describes provenance of a certain data item as the process of its creation. The second view put the focus on the source data, which is the origin of a derived data item. In the remainder of this paper we refer to the first as *transformation provenance* and the latter as *source provenance*, according to the terminology of Glavic et al. [16].

¹ A data item is a piece of data whose provenance information is of interest to a certain stakeholder.

Another aspect is *when* provenance information is computed. One possibility is to compute the provenance information when it is needed (i.e., when data is created), which is called the *lazy* approach [26]. The opposite direction is the *eager* approach, where the provenance information is computed for each transformation of the data [26].

Finally, there is an important aspect, that is the distinction between *Why*- and *Where*-provenance [4]. With *Why*-provenance, all data items that participated in the creation of a data item are captured [11]. By contrast, *Where*-Provenance focuses on the origin of the data source, which means that we can trace back a result of a transformation chain to the original data.

2.2 Watermarking

Watermarking is an active security approach that alters a work (image, program, etc.) to embed a message [9]. Fragile watermarks are an important technique for the protection of authenticity and integrity of data. In contrast, robust watermarks are related to copyright protection. For our purposes, two kinds of watermarking are of interest: Firstly, *database watermarking*, because we want to ensure trustworthiness of provenance data in databases. Second, *digital watermarking*, because the data that is subject to our watermarking approach are multimedia data items. Although the basic processes of both approaches are similar, they differ in the underlying data that is subject to the watermarking process.

In database watermarking, the *whole* database is subject to the watermarking process. As a result, the process has to deal with a heterogeneous data basis, because the different relations (of a database) consist of independent objects and tuples. In general, database watermarking consists of two steps, *watermark insertion* and *watermark detection*. We depict the general database watermarking process in Figure 1. For insertion, a generated key (e.g., based on a tuples primary key) is used to embed watermark information in the respective database. This produces a watermark DB that can be published. In general, the information, embedded into the watermark, is independent of the content of the DB. Rather, the applied *watermark schemes* are chosen with respect to certain characteristics, the watermark should have. Examples for such characteristics are the underlying data type [1], sensitivity to database attacks [17], the watermark information (*single-bit vs. multiple-bit*) or the verifiability of the watermark. For detection, an appropriate key and also the watermark information is needed. Based on this input, the detection process can determine whether a certain database contains a valid watermark or not. Generally, this approach is not invertible.

In digital watermarking, a single (digital) data item such as an image or other multimedia data item is subject to the watermarking process. Usually other algorithms and

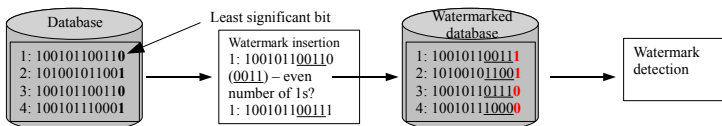


Fig. 1. General Database Watermarking Process

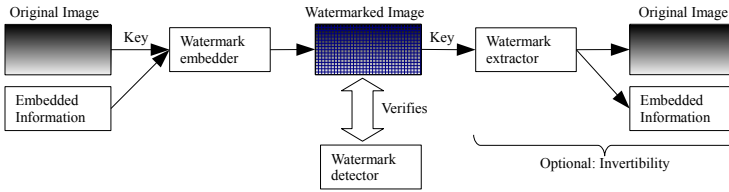


Fig. 2. General Digital Watermarking Process

concepts are used to watermark the data in digital watermarking (compared to database watermarking). We depict the general watermarking process in Figure 2. In general, a digital watermark system consists of two components: First, the *embedder*, which is responsible for embedding the watermark in the multimedia data. Second, the *detector*, which determines whether a watermark is present or not [9]. In the case that an invertible watermark is used, the system is extended by a *watermark extractor* that restores the original data. For embedding a watermark, the embedder needs three inputs: The original, unwatermarked work (i.e., the digital data) the information (data) to embed and a key. As a result, we create the watermarked data that serves as input for the detector. As with database watermarking, a digital watermark system can have different characteristics such as visibility, robustness, fragility, transparency, capacity and invertibility. The characteristics of a specific watermark system depends on the application the watermark is used for and can be chosen according to the system requirements. Amongst others, proof of ownership and content authentication are common applications of watermarking. For a detailed overview of watermark application areas we refer to Dittmann et al. [12].

2.3 Invertible Watermarking Techniques

A major problem of watermarking techniques is that watermarked data always contains a particular amount of noise (i.e., the watermark) depending on the watermarking scheme [14]. Consequently, watermarked data used in complex transformations (e.g., image quality enhancement) may lead to incorrect or at least biased results. To overcome these limitations, *invertible* watermark approaches that can extract the watermark and restore the original data have been recently proposed in the literature. According to Feng et al., there are three major approaches to embed invertible watermarks: (1) Compression based approaches, (2) Difference expansion, and (3) Histogram shifting [14].

Compression based approaches. These approaches compress some parts of the image and use the gained space to embed the watermark message.

Difference expansion. These techniques either use pixel pairs or blocks of pixels. The difference of colour values among the used pixels is expanded to create space for the embedding procedure. This procedure is invertible. Subsequently, we will explain one of these techniques in more detail.

Histogram shifting. A histogram shifting scheme first segments an image into several blocks. Then, it computes the histogram of pixel colour values for each block. To embed an a one, these techniques change the sequence of histogram bins according to a certain algorithm.



Fig. 3. Original image (left side) and applied watermarking scheme of Merkel et al. (right side) [22]

Watermarking Scheme of Merkel et al. As we implemented a variant of the watermarking scheme of Merkel et al. [22] for our motivating example, we will introduce the techniques of this scheme in more detail. Basically, this scheme uses compression and difference expansion. In Figure 3, we depict an original (on the left side) and a watermarked image of a finger print (right side) on a hard disk platter. Because this scheme was developed to ensure the privacy of a fingerprint as well as the authenticity, and integrity of the image and of the embedded data, it compresses and encrypts the pixels in the area containing the fingerprint. Furthermore, this visible² compression is used to embed parts of the watermark message. To ensure the integrity of the image and the embedded data, the scheme uses signatures and hashes (H). Particularly, the signature S is computed as $S = E(H(Image)|Message)$ and embedded into the image as well, where E is some cryptographic function. To restore the image, the retrieval phase of the scheme has to know which parts of the image are compressed and encrypted. This information (location map) is embedded into the image as well using the invisible difference expansion technique of Coltuc [8].

Difference expansion. Techniques, such as Tian's [28] and Coltuc's scheme [8] increase the difference of the colour values of pixel pairs or blocks of pixels. Since we implemented the scheme of Merkel et al. that uses Coltuc's fast embedding technique to show the feasibility of our idea, we will briefly explain this technique. To apply Coltuc's scheme, the algorithm initially splits the image into a set of pixel pairs (x, y) , according to rows, columns or any space-filling curve. Second, the algorithm performs the difference expansion of each pair using Equation 1 to compute the new pair (x', y') .

$$t(x, y) = (x', y') \quad \text{as} \quad x' = 2x - y \quad y' = 2y - x \quad (1)$$

To embed a bit of the watermark message, Coltuc's scheme modifies the transformation t of Equation 1 as shown in Equation 2. The bit of the watermark is embedded into the least significant bit (LSB) of y' . Furthermore, the scheme sets the LSB of x' to one if x and y are even; else the LSB of x' is set to zero. The marking of the LSB of the x' values is important for the inverse transformation.

$$t(x, y, w) = (x', y'), \quad w \in \{0, 1\} \quad \text{as} \quad x' = u(2x - y, x, y) \quad y' = 2y - x + w$$

$$x' = u(x_{temp}, x, y) = \begin{cases} x_{temp} \vee 1, & \text{if } x \text{ and } y \text{ even,} \\ x_{temp} \wedge \neg 1 & \text{else.} \end{cases} \quad (2)$$

² For invisible compression based schemes refer to Celik et al. [6]

Restoring the Original Image. The algorithm restores the watermark message (wm) as follows: (1) If the LSB of $x' = 1$ add the LSB of y' to wm and set the LSB of x' and y' to 0, (2) If the LSB of $x' = 0$ add the LSB of y' to wm and set the LSB of x' and y' to 1. For restoring the original image, Coltuc's scheme then performs the inverse transformation depicted in Equation 3

$$x = \lceil \frac{2}{3}x' + \frac{1}{3}y' \rceil, \quad y = \lceil \frac{1}{3}x' + \frac{2}{3}y' \rceil \quad (3)$$

3 Fingerprint Pattern Data – A Motivating Example

Fingerprints, that are traces of an impression from the friction ridges of any part of a human hand, have a long tradition in crime detection [15]. In general, the latent fingerprints detected at a crime scene are used for fingerprint pattern identification (a.k.a. *dactyloscopy*) in order to find the criminal [2].

As with today, new approaches for *contactless* latent fingerprint detection are subject to intensive research, especially in the field of signal or image processing [21][3][20]. As a result, the detected fingerprints are available in digital rather than physical form. This, in turn, leads to problems that are common for digital data, especially regarding IT security aspects like *authenticity*, *integrity* or *confidentiality*.

Furthermore, for the legal usage of digital fingerprints, we have to ensure another important aspect: the *chain-of-custody*. To this end, it is necessary to guarantee the traceability of fingerprint data throughout all transformation steps, and as a result, that each derived data item can be tracked back to the original data items. This is especially necessary if the digital fingerprint data is used as a proof at court. Here, two questions have to be answered without a doubt: Firstly, does any derived data item (used as proof) stem from the original digital fingerprint? And secondly, can we ensure the trustworthiness of the source provenance information? Unfortunately, neither standard DB mechanisms such as access control nor current data provenance approaches support us in answering these questions. Hence, we must introduce an appropriate approach, which we will sketch in the following section.

4 Using Watermarking for Data Provenance

In this section, we present an approach that allows a database system to use existing invertible watermarking schemes to a) gather provenance information for digital fingerprint data and b) ensure the trustworthiness of the provenance information. First, we present the database infrastructure in which the presented approach should be used. Second, we explain our approach and how it ensures reliability of provenance data. Finally, we show parts of the implementation with help of a simplified running example.

4.1 Architecture and General Data Flow

In Figure 4, we present an abstract overview of our DB architecture and how it is embedded in an infrastructure for fingerprint verification. The original data results from

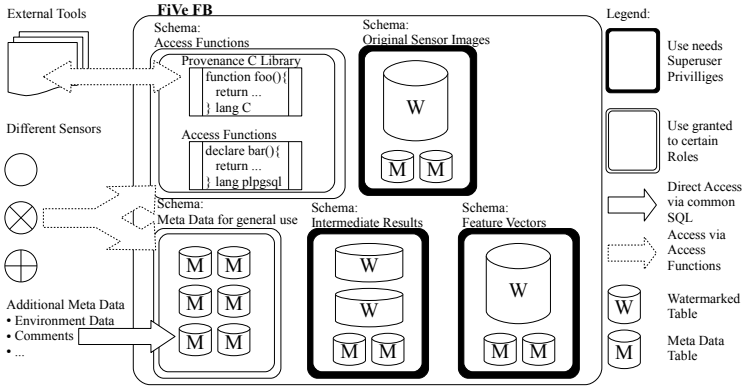


Fig. 4. Architecture

different sensors that create an image of a latent fingerprint. These images and *sensor-specific* meta data (such as resolution, sensor ID, etc.) are sent to our *Fingerprint Verification Database (FiVe DB)*. Note, that a sensor itself has no *insert* privileges on the table containing the original sensor images. Instead, a sensor calls a *user defined function* (UDF), which tests whether the image and the embedded provenance information hold integrity and authenticity (i.e., contains a valid watermark). If this is the case, the UDF stores the image and inserts the sensor-specific meta data into a separate meta data table.

Furthermore, we have a fine-grained management of meta data in FiVe DB. Meta data that is accessed frequently by different user roles such as image resolution or provenance-related meta data is stored in a dedicated schema (*Meta Data*). By contrast, meta data that is not intended for general purpose (e.g., uploader, upload time, etc.) is inserted in meta data tables in schema *Original Sensor Images*. Additionally, we collect meta data such as environmental parameters that the sensors cannot provide and store these data within schema *Meta Data* as well.

After the initial capturing of a fingerprint image, typically several transformations are performed to improve its quality or to extract some feature vectors. This is usually done by external tools or functions. To identify a certain data item (i.e., original sensor data or intermediate result due to transformation executed on the data), an external tool first queries the available meta data of this data item to get its *database id*. Next, the tool performs its transformation(s) on the data. Finally, the data item is sent back to FiVe DB together with the information on the transformation. For this new data item we have to know about the following two aspects:

- *Where* does it stem from, that is, which previous intermediate results possibly exist and what is the original image data (i.e., foreign keys to these data items),
- *How* was the new data item created (i.e., the sequence of transformations).

Since every transformation (and even initial data creation) is accompanied with a unique id of the executing unit (e.g., sensor, or tool), we can determine who changed the data. Furthermore, we obtain information regarding the *source* provenance by

connecting an original image and all of its subsequent transformation using the respective foreign keys. Additionally, we compute provenance information for each transformation step (*eager* approach) and thus have knowledge about *transformation* provenance. However, with the described architecture, it is still possible to manipulate provenance information in an unauthorised way. Hence, we introduce an approach that ensures integrity and authenticity of provenance information as well as image data in the next section.

4.2 A Hybrid Watermarking Approach

Because of our specific requirements and system infrastructure (i.e., external transformation of data), we need a watermark system that is a combination of both, database and digital watermarking. Using only a digital watermarking system would imply that the DB has neither the control nor any information on the transformation of data. Furthermore, we would not be able to perform queries to obtain (provenance) information that is possibly encoded in the watermark. By contrast, only using a database watermarking system means that the whole database has to be watermarked, which is not useful for our purposes and thus implies an overhead. Furthermore, to the best of our knowledge, no approach exists for database watermarking on unstructured or multimedia data. In the remaining section, we introduce our approach and explain, how it can be used in our secure database infrastructure.

The Hybrid Approach. To understand what makes our approach hybrid, we illustrate the insertion of a new sensor image into FiVe DB in Figure 5. The process starts with a new digital fingerprint image including embedded meta data (containing provenance information) from a sensor as visualised in Figure 1. Immediately after image creation, a *digital* watermark is embedded into the image. Therefore, the watermark is generated with a key that allows the verification of the watermark. We illustrate the watermarked sensor image by a sealed envelope.

In the second step (cf. Figure 2), FiVe DB uses the key to check, whether the watermark exists. For instance, if the data was tampered during transportation, the (fragile) watermark will be destroyed. Hence, the database can identify data items that do not hold integrity and deny their storage. After the verification of the watermark, FiVe DB

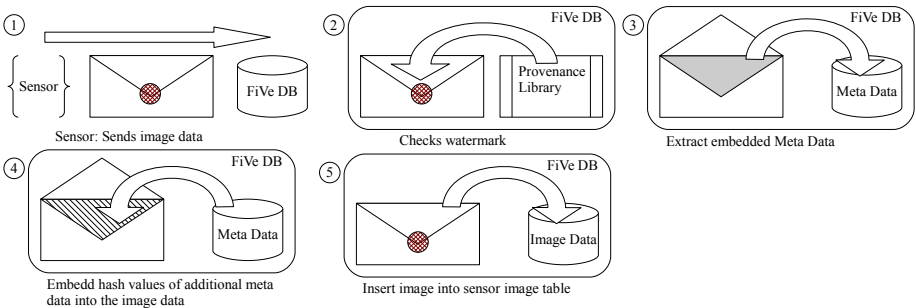


Fig. 5. Initial storage of image data

extracts the included meta data (Figure 5.3) and stores them in the database as explained in Section 4.1, so that it is accessible with common SQL. In step four (Figure 5.4), the meta data already present in FiVe DB such as environmental data are embedded into the digital image, additionally to the sensor-specific meta data. Due to limited and varying embedding capacity, (see Section 2.2) we cannot embed the whole meta data into the image. Consequently, we do not embed the data itself, but a cryptographic hash of the byte values (e.g., for strings it is just the concatenation) of each meta data item. Finally, FiVe DB seals the image data using its own key and stores it in the database.

4.3 Verifying the Watermark Information

Besides embedding the watermark, the verification of the watermark (i.e., the presence and integrity of a watermark in a data item) is an important issue. In Figure 6 we present three possible approaches for this verification step. The first possibility is that *only* the client application checks the watermark. In this case, the database can contain data that does not hold integrity (i.e., is not watermarked). The advantage of this solution is that the whole computation effort for verifying and embedding the watermark can be done on client site and thus does not affect the database performance. However, in our scenario, this approach is not suitable since there is no guarantee for the integrity of the fingerprint data and the embedded provenance information. In contrast to this approach, with the two remaining alternatives the database *additionally* checks the embedded watermark.

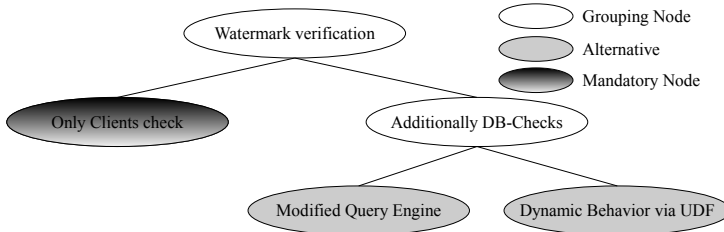


Fig. 6. Alternatives for watermark verification

The second approach uses a *modified query engine*. Such an engine verifies the watermark within the image data whenever a query on these data is performed, even when some user tries to access image meta data. The advantage of this solution is that this process is totally transparent and that the circumvention of this methodology is quite hard, because you need to tamper with the query engine itself. Despite this, we see serious disadvantages of this approach. First, it will take a great effort to modify an existing query engine to fulfill our requirements and verify its correct functionality. Second, we see serious performance problems when verifying the watermark in each query that addresses an image data item or corresponding meta (provenance) data.

Due to these drawbacks, we prefer the third approach. With this approach, the insertion and reading of image data, including the watermark verification, is done by *user*

defined functions (UDF). Except for the UDFs, no access via insert, update, or delete operations is granted on tables of the original sensor image schema, which contains the image as well as the provenance data. As a result, we prevent the contained data from unauthorised modification and thus, can ensure integrity and reliability. Obviously, the performance overhead of this approach is smaller compared to the *modified query engine* approach. Unfortunately, this alternative is not as transparent as the approach above and a user who gains unauthorised access to the underlying DB can (maliciously) modify the system behavior, e.g., by altering the UDFs. Currently, we use best practices to prevent the circumvention of our system. Amongst others, we restricted the administrative access to FiVe DB to one virtual user. Furthermore, the password for this user consists of two parts kept by two different person (four eyes principal). Additionally, we plan to use periodic jobs that check the integrity of the data within FiVe DB when there is little load on the system.

4.4 Extension of the Watermarking Scheme of Merkel et al.

In our infrastructure, we use the watermarking scheme of Merkel et. al. [22], which is a combination of compression based techniques and difference expansion (see Section 2.3). This scheme has two additional features that are advantageous for our purposes. The first feature is relevant for our fine grained provenance data handling (general purpose and confidential data). To this end, the watermarking scheme allows to insert a private message that can be read only with an appropriate key. Moreover, there can be an optional public part accessible without the key. Second, the scheme optionally allows to preserve the privacy of the finger print(s) in the image [22].

Watermark message format. The watermark message is a sequence of n bits: $WM = \{0, 1\}^n$. Furthermore, the message is separated into several blocks that represent different parts of the message. We show the overall structure of the watermark message in Figure 7.

Each message starts with a four byte integer showing the length of the *private message* block (l_{pr}). Particularly, $(l_{pr} + 1)$ is the first bit of another four byte integer representing the length of the *public message* block. The encrypted private message consists of two sub blocks: (1) the Prove Set containing confidential provenance information and (2) a signature (S) ensuring authenticity and integrity of the image itself *and* of the confidential provenance information in the Prove Set. Moreover, the message consists of another block containing the location map (see Section 2.3).

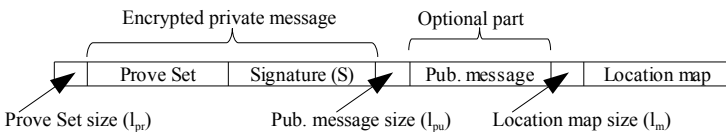


Fig. 7. Watermark message format

Increasing the Performance of Provenance Queries with redundant Prove Set storage. A Prove Set (PS) formally consists of n elements, where each element is a triple: $PS = \{(name, type, size)^n\}$. Each element consists of a name, a type (e.g., char or int), and the size of this element (in bytes). We use this structure, because it is highly related to the relational data model and can be easily extracted into (provenance) data tables. These tables *redundantly* contain all Prove Sets of the same type and thus, these Prove Sets are accessible with common SQL. Consequently, we can increase the performance of provenance queries, because the Prove Set does not have to be extracted in every query again, which is extremely costly in range queries. Note, that we can check the redundant information in the tables with help of the watermarked image at any time.

4.5 Implementation of the Hybrid Approach with Postgres 8.4

We implemented a first version of the reliable provenance approach that is based on Postgres 8.4. Although this version is yet in a premature state, we will explain the main characteristics of this approach by means of the running example.

Rights extension. Basically, we have to create a possibility that checks the presence of the watermark in an image and inserts this image, created by a sensor, initially in the database. However, since there is no write access to the image data table for the sensor, the respective user rights have to be extended *temporarily* while executing the UDF. Permanent insert privileges are not applicable, because this could lead to circumvention of the watermark checking.

Generally, two variants exist for defining the security level of an UDF in Postgres³. First, we can define the privileges of the user invoking an UDF (SECURITY INVOKER). Second, we can define (and change) the privileges of the current user to those of the user who defined an UDF (SECURITY DEFINER). However, redefining SECURITY DEFINER to extend the users privileges temporarily fits best to our needs (see Figure 8 line 22). As a result, the user associated with the sensor, can insert data into the sensor image table.

Internal Access and external Provenance Library. The UDF we use for image data insertion consists of two parts so that we can separate the *access* and *provenance* functionality. The first part contains only functions that are part of our *access library* and that are responsible for data insertion only. By contrast, the second part of the UDF contains several functions that realise meta data extraction and insertion. These functions belong to the *provenance library* and are called from access functions.

For instance, the function *insert_sensor_image* in Figure 8 is part of the access library. This function is called by a client program and executes the insertion of a new fingerprint image into FiVe DB. At the same time, it performs the privilege extension (Figure 8 line 22). Furthermore, this access function calls several functions from the *provenance library*, listed below:

- *validate_watermark()* - This function determines whether the data provided as argument contains a valid watermark.

³ <http://www.postgresql.org/docs/8.4/static/sql-createfunction.html>

```

1 FUNCTION insert_sensor_image(sensor_image bytea)
2 RETURNS integer AS $$
3 DECLARE error_code integer;
4 hash_sum character varying(512);
5 BEGIN
6 error_code = validate_watermark(sensor_image);
7 IF (error_code < 0) THEN
8     RETURN error_code;
9 END IF;
10 hash_sum = extract_meta_data(sensor_image);
11 IF (hash_sum != '') THEN
12     --USER UNDO, no Automated Rollback
13     RETURN -4;
14 END IF;
15 error = insert_image_into_repository(sensor_image,hash_sum);
16 IF (error < 0) THEN
17     --USER UNDO, no Automated Rollback
18     RETURN error;
19 END IF;
20 RETURN 0;
21 END;
22 $$ LANGUAGE 'plpgsql' VOLATILE SECURITY DEFINER
23 -- Set a secure search_path: trusted schema(s)
24 SET search_path = access_functions, pg_temp;
25
26 GRAND EXECUTE ON insert_sensor_image to role_data_collector;

```

Fig. 8. Example definition of access procedure (written in `pgsql`)

- *extract_meta_data()* - Extracts the meta data from the image data and returns the hash sum of the additional meta data (e.g., environmental data).
- *insert_image_into_repository()* - Inserts the sensor image data with the embedded hash sum into the sensor image table.

Unfortunately, it is not possible to embed transactions into a Postgres UDF, because the whole function is encapsulated into a transaction itself. Hence, we have to undo all modification manually in cases of failure instead of simply calling *rollback*.

5 Discussion

Although the presented approach is prototypically implemented, it is worth to shed light on main characteristics and open issues. Hence, we want to discuss advantages and disadvantages that such an approach may have, independent of its specific realisation.

5.1 Advantages

Reliable Provenance Information. The approach addresses authenticity and integrity of provenance data with a certain residual risk. Nevertheless, by any means we can detect provenance data that does not hold integrity due to unauthorised modifications (e.g during transportation). Additionally, it is possible to identify data that is not authentic (i.e. does not contain a valid watermark).

Resource consumption: The watermark payload is embedded directly into the image-data, raising the entropy of the data while leaving the size of the object constant. Therefore, there is no storage overhead for the watermarked data object in comparison to the unwatermarked object and the watermark is tightly coupled with the original data (see [9]). Hence, the proposed approach is efficient regarding storage requirements. Additionally, there is no additional effort for querying the embedded provenance data *directly*, since with our approach, this data is stored in separate meta data tables.

Extensibility/Adaptability: Initially, the embedded watermark contains information such as source provenance. However, in future there may be requirements for additional information regarding provenance or other issues. With our approach, the extension of the embedded watermark information is no problem. If we intend to embed additional information in our watermark message, the (watermark) system is easy to adapt so that the watermark contains the required information. We only have to take care that the watermark still fits into the original image data.

5.2 Disadvantages

Effort for infrastructure: As already mentioned, we need a holistic security infrastructure including a predefined tool chain to make our approach work. This, in turn, implies a high effort to establish such infrastructure and as a result, leads to a proprietary, overall system. However, because of the high sensitivity of the managed data such a closed system is inevitable from our point of view.

Residual risk: There exists a residual risk of circumventing the reliable provenance system. For instance, the keys for watermark generation may get lost. As a result, an unauthorised user can create data with valid watermarks and insert them into FiVe DB. Additionally, the proposed architecture also inherits some residual risks. For instance, an attacker could maliciously change the UDF definitions if he gets the password of the defining user or identify some weaknesses in Postgres itself (or the administration of the DBMS). Consequently, we intend to integrate the watermarking process into the DBMS itself, so that a DB user cannot change this behavior.

Insert and Update Performance: As we mentioned, we intend to update the watermark after each transformation. Since the update of the watermark includes its detection and (re-)insertion it may lead to a decrease of the performance overall system. Since this is not useful, we must find countermeasures to mitigate this effect. One possibility could be that result computation and update provenance information are performed in an asynchronous manner. For instance, the result could be computed and presented prior to the update of the provenance information.

6 Related Work

Data Provenance is an active field of research. With respect to databases, different approaches for data provenance exist such as annotation or inversion [35][10]. However, all approaches are applicable to (semi)-structured data only and beyond that, do not consider the reliability of provenance information. In other domains such as service-oriented architectures, research is already focused on reliability of provenance information [29]. However, the proposed concepts are often tailored to the specific domain and thus different to our approach and not applicable to the database domain.

Furthermore, a huge amount of research has been done in watermarking. For databases, different approaches have been presented, that differ in the underlying data types, subject to watermarking [1][24]. Furthermore, Sion proposes to use DB watermarking for copyright protection [25]. However, all approaches have in common that

they work on structured or at most semi-structured data and that the whole database has to be watermarked. By contrast, our approach aims at watermarking unstructured data sets that are only a subset of the whole database. In the same way as with database watermarking, digital watermarking is an active field of research that aims at information hiding in digital data such as image or audio data. Numerous watermark systems exist for embedding the watermark in subject data, depending on different requirements to the watermarked data (e.g., capacity, robustness or confidentiality) [9,19]. Furthermore, different applications can be supported by different watermarking approaches such as proof of ownership or content authentication. However, to the best of our knowledge, no approach exists where digital watermarking is integrated in a database system, neither for data protection nor provenance reliability issues.

7 Conclusion

Data Provenance gained more and more attention in the recent past and is expected to do so in future. In this paper, we proposed to apply data provenance even for unstructured data, which represents a new field of research. Moreover, we suggest to put the focus on the trustworthiness of provenance information. We presented a real-world scenario where trustworthy source provenance information is an important aspect. Subsequently, we proposed an approach how watermarking could be used to achieve both, provenance of unstructured data and its trustworthiness. Finally, we pointed out possible advantages and disadvantages of such an approach.

In the near future, we focus on finding solutions for some of the mentioned disadvantages. In detail, we search for mechanisms that increase performance for inserting/updating the watermark. Furthermore, alternative approaches (w.r.t. the current solution) for making the provenance information accessible via SQL are subject to future research.

References

1. Agrawal, R., Kiernan, J.: Watermarking relational databases. In: Proc. Int. Conf. on Very Large Data Bases, pp. 155–166 (2002)
2. Ashbaugh, D.: Ridgeology – modern evaluative friction ridge identification. *Journal of Forensic Identification* 41(1), 16–64 (1991)
3. Buneman, P., Chapman, A., Cheney, J.: Provenance management in curated databases. In: Proc. Int. Conf. on Management of Data, pp. 539–550. ACM, New York (2006)
4. Buneman, P., Khanna, S., Tan, W.C.: Why and where: A characterization of data provenance. In: Van den Bussche, J., Vianu, V. (eds.) *ICDT 2001*. LNCS, vol. 1973, pp. 316–330. Springer, Heidelberg (2000)
5. Buneman, P., Khanna, S., Tan, W.C.: On propagation of deletions and annotations through views. In: Proc. Symp. on Principles of Database Systems, pp. 150–158. ACM, New York (2002)
6. Celik, M., Sharma, G., Tekalp, A., Saber, E.: Lossless generalized-lsb data embedding. *IEEE Transactions on Image Processing* 14(2), 253–266 (2005)
7. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: Why, how, and where. *Foundations and Trends in Databases* 1(4), 379–474 (2009)

8. Coltuc, D., Chassery, J.M.: Very fast watermarking by reversible contrast mapping. *Signal Processing Letters* 14(4), 255–258 (2007)
9. Cox, I., Miller, M., Bloom, J.: *Digital Watermarking: Principles and Practice*. Morgan Kaufmann, San Francisco (2001)
10. Cui, Y., Widom, J.: Practical lineage tracing in data warehouses. In: *Proc. Int. Conf. on Data Engineering*, pp. 367–378. IEEE, Los Alamitos (2000)
11. Cui, Y., Widom, J., Wiener, J.: Tracing the lineage of view data in a data warehousing environment. *ACM Trans. on Database Systems* 25(2), 179–227 (2000)
12. Dittmann, J., Wohlmacher, P., Nahrstedt, K.: Using cryptographic and watermarking algorithms. *IEEE Multimedia* 8(4), 54–65 (2001)
13. Dubey, S., Anna, T., Shakher, C., Mehta, D.: Fingerprint detection using full-field swept-source optical coherence tomography. *Applied Physics Letters* 91(18), 1–3 (2007)
14. Feng, J., Lin, I., Tsai, C., Chu, Y.: Reversible watermarking: Current status and key issues. *International Journal of Network Security* 2(3), 161–171 (2006)
15. Galton, F.: *Fingerprints*. MacMillan and Co., NYC (1892)
16. Glavic, B., Dittrich, K.: Data provenance: A categorization of existing approaches. In: *GI-Fachtagung für Datenbanksysteme (BTW)*, pp. 227–241 (2007)
17. Guo, H., Li, Y., Jajodia, S.: Chaining watermarks for detecting malicious modifications to streaming data. *Inf. Sci.* 177(1), 281–298 (2007)
18. Gupta, A.: Data provenance. In: *Encyclopedia of Database Systems*, p. 608. Springer, Heidelberg (2009)
19. Hartung, F., Kutter, M.: Multimedia watermarking techniques. *Proc. of IEEE* 87(7), 1079–1107 (2002)
20. Kuivalainen, K., Peiponen, K.E., Myller, K.: Application of a diffractive element-based sensor for detection of latent fingerprints from a curved smooth surface. *Measurement Sci. and Tech.* 20(7), 207–211 (2009)
21. Lin, S., Engheta, N., Pugh Jr., E., Yemelyanov, K.: Polarization- and specular-reflection-based, non-contact latent fingerprint imaging and lifting. *Journal of Opt. Soc. of America* 23(9), 2137–2153 (2006)
22. Merkel, R., Kraetzer, C., Dittmann, J., Vielhauer, C.: Reversible watermarking with digital signature chaining for privacy protection of optical contactless captured biometric fingerprints - a capacity study for forensic approaches. To Appear in *Int'l. Conf. on Digital Signal Processing* (2011)
23. Simmhan, Y., Plale, B., Gannon, D.: A survey of data provenance techniques. Tech. Rep. IUB-CS-TR618, Department of Computer Science, Indiana University, Bloomington (2005)
24. Sion, R.: Proving ownership over categorical data. In: *Proc. Int. Conf. on Data Engineering*, pp. 584–595 (2004)
25. Sion, R.: Database watermarking for copyright protection. In: Gertz, M., Jajodia, S. (eds.) *Handbook of Database Security*, pp. 297–328. Springer, US (2008)
26. Tan, W.C.: Research problems in data provenance. *IEEE Data Eng. Bull.* 27(4), 42–52 (2004)
27. Tan, W.C.: Provenance in databases: Past, current, and future. *IEEE Data Eng. Bull.* 32(4), 3–12 (2007)
28. Tian, J.: Reversible data embedding using a difference expansion. *Circuits and Systems for Video Technology* 13(8), 890–896 (2003)
29. Tsai, W., Wei, X., Chen, Y., Paul, R., Chung, J.Y., Zhang, D.: Data provenance in soa: Security, reliability, and integrity. *Service Oriented Computing and Applications* 1, 223–247 (2007)

ECOS: Evolutionary Column-Oriented Storage

Syed Saif ur Rahman, Eike Schallehn, and Gunter Saake

Faculty of Computer Science,
Otto-von-Guericke University, Magdeburg, Germany
{srahman,eike,saake}@ovgu.de

Abstract. As DBMS has grown more powerful over the last decades, they have also become more complex to manage. To achieve efficiency by DBMS tuning is nowadays a hard task carried out by experts. This development inspired the ongoing research on self-tuning to make DBMS more easily manageable. We present a customizable self-tuning storage manager, we termed as Evolutionary Column-Oriented Storage (ECOS). The capability of self-tuning data management with minimal human intervention, which is the main design goal for ECOS, is achieved by dynamically adjusting the storage structures of a column-oriented storage manager according to data size and access characteristics. ECOS is based on the Decomposed Storage Model (DSM). It supports customization at the table-level using five different variations of DSM. ECOS also proposes fine-grained customization of storage structures at the column-level. It uses hierarchically-organized storage structures for each column, which enables autonomic selection of the suitable storage structure along the hierarchy using an evolution mechanism (as hierarchy-level increases). Moreover, for ECOS, we proposed the concept of an evolution path that provides a reduction of human intervention for database maintenance. We evaluated ECOS empirically using a custom micro benchmark showing performance improvement.

Keywords: column-oriented storage, evolving hierarchically-organized storage structures, customization, autonomy.

1 Introduction

Efficient data management demands continuous tuning of a database and a DBMS. The need for tuning a DBMS is driven by changes, such as database size, workloads, schema design, hardware, and application specific data management needs. Existing DBMS need extensive human intervention for tuning, which contributes to a major portion of the total cost of ownership for data management [7]. Self-tuning is the solution to reduce the tuning cost through minimizing the human intervention [22]. However, researchers are united on one conclusion that the self-tuning based solutions are the biggest challenge in the database domain because of the inherent complexity of existing DBMS architectures. Their functionalities are tightly integrated into their monolithic engines, and it is difficult to assess the impact of tuning of one knob on another [6].

In this paper, we present a customizable and online self-tuning storage manager. As a key design concept, we propose the selection of an appropriate storage

model and data/index storage structure through customization. This design decision is according to the suggestion from the work of Chaudhuri and Weikum [6]. ECOS supports fine-grained customization at the table-level and column-level according to the recommendations/results from [2,10,12]. We also identified the need to autonomically change the existing data and index storage structure to more appropriate ones with the changing data management needs according to our previously published results in [18]. We named our solution Evolutionary Column-Oriented Storage (ECOS), which is based on the existing Decomposed Storage Model (DSM) [10]. It uses hierarchically-organized storage structures for each column with an innovative evolution mechanism, which enables autonomic selection of the most suitable storage structure along the hierarchy (as the levels of a hierarchy increase). Furthermore, we present four possible variations to standard 2-copy DSM to reduce its high storage requirement. We evaluated ECOS empirically using the custom micro benchmark and our results show that ECOS self-tunes the storage structure while maintaining the required performance. Additionally, it also gives minor performance gains. Furthermore, we propose a mechanism called evolution path to define the storage structure evolution, which reduces the need for human intervention for long-term database maintenance.

This paper is organized as follows. Section 2 defines the problem and justifies the motivation for the proposed design. Section 3 explains the concepts of ECOS and evolution path in detail. Section 4 introduces the prototype implementation and gives details of the empirical evaluation of the proposed concepts using a custom micro benchmark. Section 5 outlines the related work. Section 6 concludes the paper with hints for the future work.

2 Problem Statement and Motivation

Specific storage structures have characteristics suitable for certain data sizes and access patterns. As both of these aspects may change over the course of data usage, there is no single storage solution that provides optimal performance in every situation. Therefore, we propose an autonomic adjustment of the storage structures. In this section, we explain the motivation for some critical design decisions in ECOS. To explain the problem in detail, we take the LINEITEM table of the TPC BenchmarkTMH (TPC-H) [17] schema as an example. We generated the benchmark data with the scale factor of one and gathered statistics for the LINEITEM table as shown in Table 1.

Why column-oriented storage model? The column-oriented storage model is derived from earlier work of DSM [10]. DSM is a transposed storage model [4] that stores all values of the same attribute of the relational conceptual schema relation together [10]. Copeland and Khoshafian in [10,20] concluded many advantages of DSM including simplicity (Copeland and Khoshafian related it to RISC [16]), less user involvement, less performance tuning requirement, reliability, increased physical data independence and availability, and support of

heterogeneous records. These advantages give strong motivation for the use of the DSM in a self-tuning storage manager.

Why customization at the column-level? Table 1 includes some characteristics of the LINEITEM table. We can observe that distinct data count (cardinality) for all columns is different. We further looked into the TPC-H queries that access the LINEITEM table (general observation) and predicted (using a layman-approach) the workload and data access patterns for columns. We identified that four columns involve read-intensive workload and three columns involve ordered data access as shown in Table 1. The differences in distinct data count, workload, and data access pattern for different columns raise the need for the support of storage structure customization at the column-level. If a storage manager supports the column-level customization of storage structures, we can hypothetically customize the LINEITEM table columns as shown in Table 1.

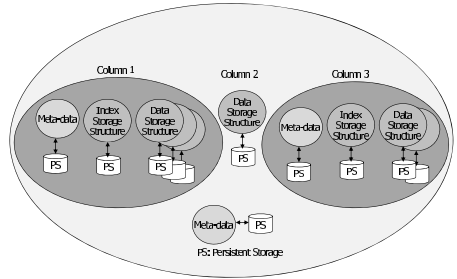
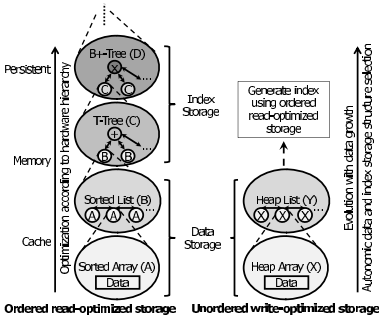


Fig. 1. Evolving hierarchically-organized storage structures

Fig. 2. Evolutionary column-oriented storage

Why hierarchically-organized storage structures? A hierarchical organization of storage structures is a composition of similar or different storage structures in a hierarchy as depicted in Figure 1. Hierarchically-organized storage structures provide an opportunity for autonomic selection of appropriate storage structures along the hierarchy. We suggest that a new storage structure will be appropriate because we can use the existing data and gathered statistics during previous operations on existing storage structures to make better decisions for the next appropriate storage structure selection. The usage of hierarchically-organized storage structures is also motivated by the possible optimization of the storage structure hierarchy according to a hardware hierarchy and data management needs. For example, consider the memory hierarchy in the modern hardware. We optimize storage structures for cache, main memory, and persistent storage in the specified order. As shown in Figure 1, the lowest level of hierarchy is using arrays, which are optimized for cache. On the second level above, T-Trees are used, which are optimized for main memory. At the third level, B+-Tree is used, which is optimal for persistent storage. Previously published results from Bender et al. [5], Chen et al. [8], and Morzy et al. [14] also influenced our decision for the use of hierarchically-organized storage structures.

Table 1. TPC-H LINEITEM table observed statistics, possible customization, and anticipated evolution

Column Name	Distinct Count	Workload	Data Access	Storage Structure Initial	Storage Structure 1st Evolution	Storage Structure 2nd Evolution
L_ORDERKEY	1500000			Sorted Array	Sorted List	B+-Tree
L_COMMENT	4501941			Sorted Array	Sorted List	Hash Table
L_DISCOUNT	11	Read-Intensive		Sorted Array		
L_SHIPMODE	7			Heap Array		
L_SHIPINSTRUCT	4			Heap Array		
L_RECEIPTDATE	2554			Heap Array	Heap List	
L_COMMITDATE	2466		Ordered	Sorted Array	Sorted List	
L_SHIPDATE	2526		Ordered	Sorted Array	Sorted List	
L_LINESTATUS	2			Heap Array		
L_RETURNFLAG	3			Heap Array		
L_TAX	9	Read-Intensive		Sorted Array		
L_EXTENDEDPRICE	933900	Read-Intensive		Sorted Array	Sorted List	B+-Tree
L_QUANTITY	50	Read-Intensive	Ordered	Sorted Array		
L_LINENUMBER	7			Heap Array		
L_SUPPKEY	10000			Heap Array	Heap List	
L_PARTKEY	200000			Sorted Array	Sorted List	Hash Table

3 Evolutionary Column-Oriented Storage

In this section, we explain the concepts of ECOS in detail. We introduce and explain four DSM based schemes proposed to reduce the high storage requirement of standard 2-copy DSM. We also discuss the concepts of the table and the column customization, hierarchical organization and evolution of the storage structures, and the evolution path.

3.1 Table-Level Customization

ECOS is a customizable and online self-tuning storage manager. We use the term storage manager in its standard meaning for DBMS, i.e., a component to physically store and retrieve data. Data storage efficiency is assumed to be the main goal for a storage manager. By storage structure, we mean the data structure used by the storage manager to physically store data and indexes. ECOS stores data according to the column-oriented storage model, where each column stores a key/value pair of data. ECOS suggests two customizations for each table in a database, i.e., at the table-level and at the column-level. At the table-level, we customize, how columns are stored physically for a logical schema design. We use five variations of DSM for table customization, i.e., Standard 2-copy DSM [10], Key-copy DSM (KDSM), Minimal DSM (MDSM), Dictionary based Minimal DSM (DMDSM), and Vectorized Dictionary based Minimal DSM (VDMDSM). The motivation for proposing and testing different variations of

Table 2. DSM

Columnk0		Columnk1		Columnk2		Columnv0		Columnv1		Columnv2	
Key	Value	Key	Value	Key	Value	Key	Value	Key	Value	Key	Value
k1	731	k1	20090327	k1	Jana	k2	137	k3	20010925	k3	Christian
k2	137	k2	20071201	k2	Tobias	k3	173	k6	20010925	k1	Jana
k3	173	k3	20010925	k3	Christian	k5	317	k2	20071201	k6	Jana
k4	371	k4	20090327	k4	Tobias	k4	371	k1	20090327	k2	Tobias
k5	317	k5	20090327	k5	Tobias	k6	713	k4	20090327	k4	Tobias
k6	713	k6	20010925	k6	Jana	k1	731	k5	20090327	k5	Tobias

(a) Columns clustered on key

(b) Columns clustered on value

Table 3. MDSM

Columnk1		Columnk2		Columnv0	
Key	Value	Key	Value	Key	Value
k1	20090327	k1	Jana	k2	137
k2	20071201	k2	Tobias	k3	173
k3	20010925	k3	Christian	k5	317
k4	20090327	k4	Tobias	k4	371
k5	20090327	k5	Tobias	k6	713
k6	20010925	k6	Jana	k1	731

(a) Columns clustered on key

(b) Primary key columns clustered on value

DSM arise from high storage requirement of standard 2-copy DSM. The details for the five variations of DSM are as follows:

Standard 2-copy DSM. DSM is a transposed storage model [4], which pairs each value of a column with the surrogate of its conceptual schema record as key [10]. It suggests storing two copies of each column, one copy clustered on values, whereas another copy is clustered on keys. DSM is depicted in Table 2. We argue that for a self-tuning storage manager, 2-copy DSM is the most suitable storage model. It is easy to implement and easy to use, moreover, it does not require human intervention to identify which column to cluster or index, instead it is done in a uniform way [20]. To justify our argument, we evaluated standard 2-copy DSM with four other variations and found it the most appropriate one. The results are presented in Section 4.

Key-copy DSM (KDSM). KDSM is the first variation of DSM that we propose to reduce the high storage requirement of the standard 2-copy DSM. KDSM stores the data similar to DSM, i.e., for each column, data is stored in values, whereas keys are unique numeric values that relate attributes of a row together. All columns are clustered on the keys. However, unlike DSM, we store an extra copy of only key columns (primary key or composite primary key) clustered on values. This design alteration reduces the storage requirement of KDSM, but it increases the access time for read operations that involve non-key columns in search criteria. However, for read operations with the key column in the search criteria it performs similar to DSM with less storage requirement as shown in Section 4. We propose the use of KDSM for tables that only require querying data using key columns.

Minimal DSM (MDSM). MDSM stores the data similar to DSM except that we do not store any extra copy for any columns thus reducing the high storage requirement of DSM to a minimum. Instead, the design idea of MDSM is to store primary key columns clustered on values, whereas non-primary key columns are clustered on key as depicted in Table 3. MDSM performs similar to DSM and KDSM for the read operations with search criteria on key column attributes, but it performs worse for the read operations with non-key column attributes

Table 4. Dictionary columns for DMDSM and VDMDSM

Dict. Column 0		Dict. Column 1		Dict. Column 2	
Keyd0	Valued0	Keyd1	Valued1	Keyd2	Valued2
d02	137	d11	20090327	d23	Christian
d03	173	d12	20071201	d21	Jana
d05	317	d13	20010925	d22	Tobias
d04	371				
d06	713				
d01	731				

(a) Dictionary columns

Table 5. DMDSM

Columnv0		Columnk1		Columnk2	
Keyv0	Valuev0	Key	Value	Key	Value
k2	d02	k1	d11	k1	d21
k3	d03	k2	d12	k2	d22
k5	d05	k3	d13	k3	d23
k4	d04	k4	d11	k4	d22
k6	d06	k5	d11	k5	d22
k1	d01	k6	d13	k6	d21

(a) Primary key columns (b) Columns clustered on key clustered on value

Table 6. VD-MDSM

Vector Column	
Key	Value
v1	d01,d11,d21
v2	d02,d12,d22
v3	d03,d13,d23
v4	d04,d11,d22
v5	d05,d11,d22
v6	d06,d13,d21

(a) Vector column

in search criteria as shown in Section 4. Our results in Section 4 suggest that if we do not have any space constraint and we do need access using non-key attributes, this scheme is not appropriate.

Dictionary based Minimal DSM (DMDSM). To improve the performance of MDSM, we introduced DMDSM, which stores the unique data for each column separately as the dictionary column. DMDSM is inspired from the concept of the dictionary encoding scheme, which is frequently used as light-weight compression technique in many column-oriented data management systems [1]. In DMDSM, for each main column, values are the keys for the data from dictionary column as depicted in Table 5. All dictionary columns are clustered on value. All other concepts for the DMDSM are similar to MDSM. This scheme gives us the provision to exploit our innovative concept of evolving hierarchically-organized storage structures to its maximum potential for dictionary columns.

Vectorized Dictionary based Minimal DSM (VDMDSM). VDMDSM is an extension of DMDSM, such that it stores the values (i.e., dictionary column keys) for all columns together as the vector column, i.e., instead of saving each column separately, it generates the vector of all attributes in the row and stores it as a value for vector column as depicted in Table 6. Similar to DMDSM, VDMDSM provides the opportunity to exploit the benefit of evolving hierarchically-organized storage structures to their full potential for dictionary columns.

3.2 Column-Level Customization and Storage Structure Hierarchies

Once we select the appropriate storage model scheme from above-mentioned schemes at the table-level, we move forward to customize the columns as explained next. At the column-level, we customize the storage structure for each column. Each column is initially customized as either ordered read-optimized or unordered write-optimized storage structure. For ordered read-optimized storage structures, we store data in sorted order with respect to key or value, whereas for unordered write-optimized storage structure, we store data according to insertion order. In the above-mentioned schemes, dictionary columns are always stored as ordered read-optimized storage structures.

Evolving hierarchically-organized storage structures. ECOS utilizes the hierarchically-organized storage structure for data and index storage, such that a storage structure at each new level of hierarchy is composed of multiple lower level storage structures as depicted in Figure 1. The storage structures that we discuss in this paper include heap array, sorted array, heap list, sorted list, B+-Tree, T-Tree, and hash table. Before we continue our discussion, we outline the hierarchically-organized storage structures, which we use further in our discussion. At the lowest level of hierarchy, we used:

Sorted array: Optimized for read-access with minimal space overhead. No need to instantiate a buffer manager or an index manager to manage an array.

Heap array: Optimized for write-access with minimal space overhead.

At the next level, we used composite storage structures:

Sorted list: Sorted list is composed of multiple sorted arrays. It requires the instantiation of a buffer manager for managing multiple sorted arrays.

Heap list: Heap list is composed of multiple heap arrays. It also requires the instantiation of a buffer manager for managing multiple heap arrays.

B+-Tree: B+-Tree is composed of multiple arrays as leaf nodes. It requires the instantiation of a buffer manager for managing multiple arrays as well as an index manager to manage the multiple index nodes.

On the higher levels, we used high-level composite (HLC) storage structures:

HLC SL: HLC SL is a B+-Tree based structure, where each leaf node is a sorted list. HLC SL instantiates a buffer manager to manage multiple sorted lists and an index manager to manage multiple index nodes. Each sorted list manages its own buffer manager, which ensures the high locality of data for each sorted list.

HLC B+-Tree: HLC B+-Tree is a B+-Tree based structure, where each leaf node is also a B+-Tree. HLC B+-Tree instantiates a buffer manager to manage multiple B+-Trees and an index manager to manage multiple index nodes. Each B+-Tree at leaf nodes has its own buffer manager and index manager, which ensures the high locality of data and index nodes for each B+-Tree.

Once a column is customized as either ordered read-optimized or unordered write optimized storage structure, ECOS initializes each column to the smallest possible storage structure, i.e., an ordered read-optimized column is initialized as a sorted array, whereas an unordered write-optimized column is initialized as a heap array. ECOS enforces that each storage structure should be atomic and should be directly accessible using an access API. The reason for this approach is that small storage structures consume less memory and generate reduced binary size for small data management [18]. If we can use them directly, then there is no reason to use them as part of complex storage structures¹, such as

¹ We use storage structure as a common term for both data storage structure and index storage structure.

B+-Tree or T-Tree; avoiding the overheads of complexity associated with these storage structures. This approach ensures that using smallest suitable storage structures, desired performance is achieved using minimal hardware resources for small database management.

Storage capacity limitations for predictable performance. ECOS imposes data storage capacity limitations for each storage structure. We enforce this for more predictable performance and to ensure that storage structure performance does not degrade because of unlimited data growth. In ECOS, once the limited storage capacity of a storage structure is consumed, it evolves to a larger more complex storage structure composed of multiple existing ones considering the important factors, such as hardware, the data growth, and the workload. For ordered read-optimized data storage, a sorted array is evolved into a sorted list. For unordered write-optimized data storage, a heap array is evolved into the heap list. The evolution of storage structure is an important event for assessing the next suitable storage structure by analyzing the existing data and the previously monitored workload. Similarly, each new storage structure also has a definite data storage capacity limitation and, once again, as it is consumed, ECOS further evolves and increases the hierarchy of the hierarchically-organized storage structures.

API consistency to hide complexity and ensure ease of use. To hide the complexity of different storage structures over different levels of hierarchy, ECOS keeps the interface for all storage structures consistent. We provide a standard interface to access columns with simple, Put(), Get(), and Delete() functionality with record as argument. It is invisible to an end-user, which storage structure is currently in use for each column.

Automatic partitioning. ECOS separates physical storage for each column to reduce the I/O contention for storing large databases. For large columns, it also separates the data for a column into multiple separate physical storage units, which is similar to horizontal partitioning. In Figure 2, at a minimum each column has its own separate physical storage. With the growth of data, each column may spread over multiple physical storage units. For example, for storage structures of Table 1, each sorted list or heap list will be stored in a separate data file, whereas each B+-Tree or T-Tree will be stored in a separate index file. These physical storage units may be stored on the single hard disk, or they may spread across the network.

3.3 Evolution and Evolution Paths

By evolution, we mean the transformation of a storage structure from an existing form into another form such that the previous form becomes an integral and atomic unit of the new form autonomically. Evolution path is the mechanism to define how ECOS evolves a smallest simple storage structure into a large complex storage structure. It consists of many storage structure/mutation rules pair entries that ECOS uses to identify, how to evolve the storage structures. Each

Table 7. Example for evolution paths

Storage Structure: Initial	Mutation Rules	Storage Structure: 1st Evolution	Mutation Rules	Storage Structure: 2nd Evolution
Sorted array	Event: Sorted array=Full Heredity based selection: Workload=Read intensive Data access=Unordered Mutation: => Evolve (Sorted array - >Sorted list)	Sorted list of sorted arrays	Event: Sorted list=Full Heredity based selection: Workload=Read intensive Data access=Ordered Mutation: => Evolve (Sorted list - >B+-Tree)	B+-Tree of sorted lists(As leaf nodes for data storage)
Sorted array	Event: Sorted array=Full Heredity based selection: Workload=Read intensive Data access=Ordered Mutation: => Evolve (Sorted array - >B+-Tree)	B+-Tree of sorted arrays(As leaf nodes for data storage)	Event: B+-Tree=Full Heredity based selection: Workload=Read intensive Data access=Ordered Mutation: => Evolve (B+-Tree - >HLC (B+-Tree based))	HLC of B+-Tree(As leaf nodes)
Sorted array	Event: Sorted array=Full Heredity based selection: Workload=Write intensive Data access=Unordered Mutation: => Evolve (Sorted array - >Heap array)	Heap list based on heap array mutation rules		
Heap array	Event: Heap array=Full Heredity based selection: Workload=Write intensive Data access=Ordered Mutation: => Evolve (Heap array - >Heap list) & Generate (Secondary index = Sorted list)	Heap list	Event: Heap list=Full Heredity based selection: Workload=Write intensive Data access=Ordered Mutation: => Evolve (Heap list - >Hash table) & Evolve (Secondary index = Sorted list - >B+-Tree)	Hash table
Heap array	Event: Heap array=Full Heredity based selection: Workload=Write intensive Data access=Unordered Mutation: => Evolve (Heap array - >Heap list)	Heap list	Event: Heap list=Full Heredity based selection: Workload=Write intensive Data access=Unordered Mutation: => Evolve (Heap list - >Hash table)	Hash table

storage structure can have multiple mutation rules mapped to it. These mutation rules consist of three information elements: Event, Heredity based selection, and Mutation. The event identifies, when this mutation rule should be executed. Different mutation rules can have the same event, but not all of them execute the mutation. The heredity based selection identifies precisely, when evolution should occur based on the heredity information gathered for the existing storage structure. Heredity information means the gathered statistics about the storage structure, e.g., workload type, data access pattern, previous evolution details, etc. The mutation defines the actions that should be executed to evolve the storage structure. Example of an evolution path is shown in Table 7. We envision that common DBMS maintenance best practices can be documented using the evolution path mechanism. ECOS assumes that DBMS vendors provide the evolution paths that best suit their DBMS internals, with the provision of alteration for a database administrator. The only liability for configuration that lies with database designers and administrator is to have a look at the evolution path for the DBMS and if needed, alter it with desired changes. Evolution process in

ECOS is autonomic, and it exploits evolution path to automatically evolve the storage structures, i.e., our approach for self-tuning is online.

Consider the `L_ORDERKEY` column of the `LINEITEM` table as shown in Table 1. Suppose, as a database designer, we design this table. According to our application design, we select the `L_ORDERKEY` column as a part of the primary key. As we already discussed in Section 3, we have to customize each column as either ordered read-optimized or unordered write-optimized. Therefore, we customize the `L_ORDERKEY` column as ordered read-optimized. At the initial design time, we design according to the domain knowledge, our experiences, and predictions. As a designer, it is difficult to guarantee, how much this column grows, and how long it takes to reach that size. When we customize the column as ordered read-optimized, it is internally initialized as a sorted array. Now for the `L_ORDERKEY` column, three initial rows of the sample evolution path of Table 7 are relevant.

As we mentioned in Section 3, ECOS limits the storage capacity for each storage structure. Therefore, the initial sorted array has a certain data storage capacity limit. For example, consider it as 4KB. As long as data is within the 4KB limits, sorted array is the storage structure for the `L_ORDERKEY` column, and we gather the heredity information for the column, such as the number of `Get()`, the number of `Put()`, the number of `Delete()`, the number of range `Get()` (for range queries), the number of `Get()` for all records (for scan queries), etc. What heredity information should be gathered may vary from one implementation to another. Here, we simplify our discussion by assuming that a system can identify using heredity information that the workload is either read-intensive or write-intensive and the access to data is either ordered (range queries) or unordered (point or scan queries).

The moment the storage limit of the sorted array is consumed, an event is raised for notification. This event triggers all three initial mutation rules of Table 7. Now heredity based selection identifies, which one of them to execute. We suppose that for the `L_ORDERKEY` column, the workload is read-intensive and the data access is unordered, this scenario executes the first mutation rule of Table 7, which evolves the existing sorted array into a sorted list. Now sorted list is the new storage structure, and it is also constrained with the storage limit according to the design principle of ECOS. As long as the `L_ORDERKEY` column data is within the storage limit of the sorted list, heredity information is gathered, and it is used for the next evolution.

It is observed from Table 1 that only half of the `LINEITEM` columns, i.e., eight out of sixteen with high data growth evolve during the first evolution. The rest of the columns can be stored within an array (either heap array or sorted array). Furthermore, only half of the columns with high data growth, i.e., four out of eight, which are evolved during the first evolution evolve again during the second evolution (i.e., `L_ORDERKEY`, `L_COMMENT`, `L_EXTENDEDPRICE`, and `L_PARTKEY`). The final state of the table presented in Table 1 shows that each column is using the appropriate storage structure (we assume for explanation) according to the stored data size and observed workload. We can add more parameters for evolution decisions, but we only used limited parameters

(i.e., data size, workload, and data access) to keep our discussion simple and understandable. Table 1 shows only the evolution for dictionary columns for the LINEITEM table as they utilize the benefits of evolving hierarchically-organized storage structures to their full potential. Before we conclude this section, to avoid any confusion, we want to disclaim that the terms and concepts of evolution, evolution path, mutation rules, and heredity information used in this paper have no relevance with their counterpart in evolutionary algorithms or any other non-relevant domain.

4 Implementation and Empirical Evaluation

In this section, we provide the details of our micro benchmark and the evaluation results for ECOS². The data and index storage structures that we have implemented in the existing ECOS prototype implementation are the same as we have discussed in Section 3.2. To simplify our discussion, we present the results involving sorted array, sorted list, and HLC SL.

4.1 Micro Benchmark Details

For ECOS evaluation, we set up a micro benchmark with repeated insertion, selection, and deletion of data using API based access methods. The data contain keys in ascending, descending, and random order, which also represents their insertion, selection, and deletion order in the database. For different columns, the number of records (cardinality) is kept different. We defined seven columns with two unique non-null columns, one of them used as a primary key. We used three different widths for columns, i.e., 16, 85, and 4096 bytes to assess the impact of tuple width on performance of different storage schemes. All storage structures used in a micro benchmark operate in main-memory. For ECOS evaluation, we used CPU cycles and heap memory as resources. We used OpenSuse 11.2 operating on Intel(R) Core(TM)2 Duo CPU E6750 @ 2.66GHz with four GB of RAM. We measured execution speed by taking the average of CPU cycles observed over multiple iterations of the micro benchmark. We used Valgrind tools suite [21] to measure the heap usage.

4.2 ECOS Performance Improvement

To demonstrate the performance gain using ECOS, we presented our observation of the effect of an increase in data size on performance of different storage structures in [18,19]. According to our observation in [18,19], we suggest the performance gain and reduced resource consumption using the evolving storage structures because evolving storage structures attempt to use minimal/simple storage structures (such as sorted array for small data management) as long as possible using the definitions from evolution paths. To demonstrate the evolving storage structures evolution, we present the evaluation results for evolving

² Please refer to the web link for all related publications and prototype evaluation binaries: <http://wwiti.cs.uni-magdeburg.de/~srahman/CellularDBMS/index.php>

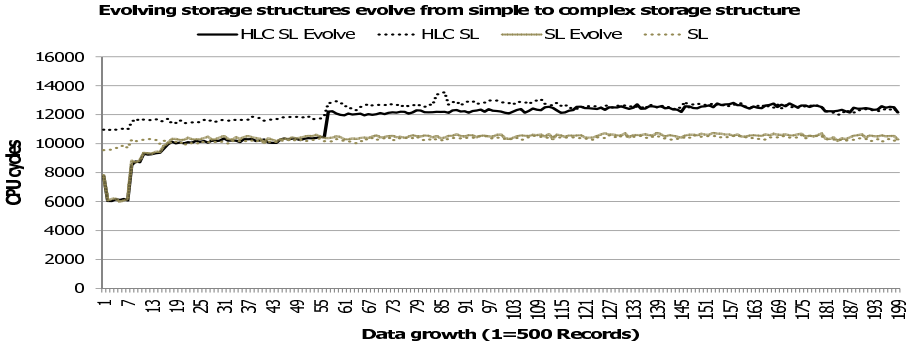


Fig. 3. Evolving HLC SL storage structure evolution

HLC SL storage structure in Figure 3 (due to space constraint the evaluation results for evolving HLC B+-Tree can be found in [19]). It can be observed in Figure 3 that the evolving storage structure HLC SL evolves with the data growth. It can be observed that the HLC SL storage structure consume more CPU cycles in comparison with sorted list and sorted array. This behavior is due to the complexity of the storage structure, which is meant to be used for extremely large data sizes. The HLC SL storage structure automatically partitions the data and uses separate buffer and index managers for each partition, which is not the requirement for presented 500K records storage. However, we forced storage structures to evolve to HLC SL for 500K records for the purpose of demonstration of the evolution concept.

In Figure 4 and 5, we present the performance comparison of different DSM based schemes that we explained in Section 3. The results in Figure 4 and 5 show that DSM and KDSM perform better for evaluation with search criteria on key-attributes, whereas for evaluation with search criteria on non-key attributes DSM outperforms the other schemes. It is observed that storage requirement for DSM is highest, whereas the storage requirement is the lowest for VDMDSM. It is

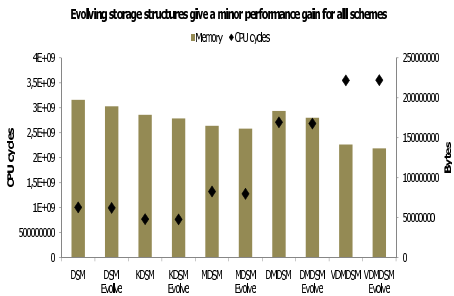


Fig. 4. Performance comparison of different DSM based schemes in ECOS with primary key based search criteria

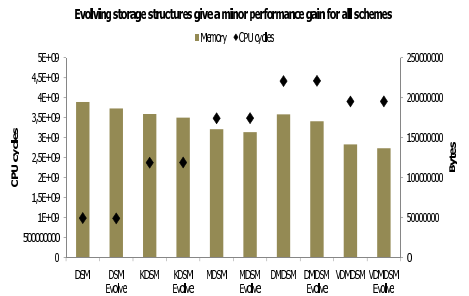


Fig. 5. Performance comparison of different DSM based schemes in ECOS with non-key based search criteria

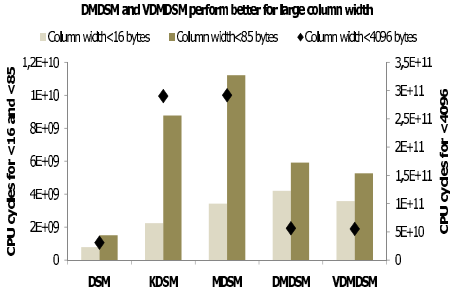


Fig. 6. Performance improvement for dictionary based DSM schemes for large column width

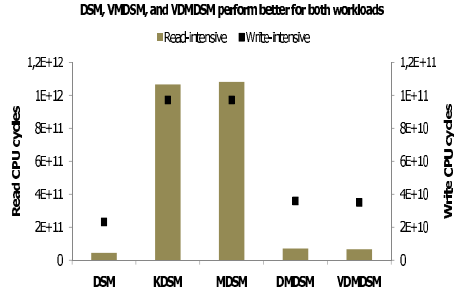


Fig. 7. Performance comparison of different DSM based schemes in ECOS for read and write intensive workloads

also observed that evolving storage structures perform better than fixed storage structures with minor performance gains. As we have discussed in Section 4, our work is based on the ideology from Chaudhuri and Weikum presented in [6]. They used the notion of “gain/pain ratio” to discuss the overall gain of their proposed approach. They advocate the ideology of less complex, more predictable, and self-tuning RISC-style components with minor compromise on performance to achieve overall improvement in “gain/pain ratio”. Our results show the minor performance gain, which should be a good achievement considering the overall benefits we achieve in terms of simplicity, predictability, and self-tuning.

It can be observed in Figure 6 that dictionary based schemes performance is improved and becomes comparable with standard 2-copy DSM scheme for large tuple width. However, KDSM and MDSM still perform poor. We also analyzed the performance difference for different DSM schemes on both the read-intensive and write-intensive workloads. It is observed in Figure 7 that for write-intensive workload DSM outperforms other schemes; however, for the read-intensive workload differences in performance between the 2-copy DSM and the dictionary based DSM schemes is minimum. This is a promising result for dictionary based schemes, and it shows their potential to act as a better alternative to 2-copy DSM after overcoming their short comings.

5 Related Work

Hierarchically-organized storage structures have already been in use in the data warehousing domain. Morzy et al. in [14] proposed a hierarchical bitmap index for indexing set-valued attributes. Later, Chmiel et al. in [9] extended that concept to present hierarchically-organized bitmap indexes for indexing dimensional data. Bender et al. proposed cache-oblivious B-Trees [5] that perform the optimal search across different hierarchical memories with varying memory levels, cache size, and cache line size. Fractal prefetching B+-Trees [8] proposed by Chen et al. are the most relevant work for the ECOS and is similar in concept to cache-oblivious B-Trees with an additional concept of prefetching. Fractal prefetching

B+-Trees are optimized for both cache and disk performance, which is also a goal for the ECOS. However, the ECOS concepts do not restrict the use of any fixed structure; instead it suggests the use of different storage structures in the hierarchy to support an efficient use of underlying hardware.

An automated tuning system (ATS) [11] is a feedback control mechanism that automatically adjusts the tuning knobs using the defined tuning policies according to the monitoring statistics. ECOS also works in similar fashion as suggested in ATS. ECOS also monitors and adjust storage structures with changing data management needs. Malik et al. in [13] suggested the benefit of online physical design techniques and proposed an online vertical partitioning technique for physical design tuning. Similarly, ECOS also operates in online fashion. Automated physical design research focuses on finding the best physical design structure for a running workload, e.g., indexes, materialized views, partitioning, clustering, and views [3]. Existing automated physical design tools assume the workload as a set of SQL statements [3]. These tools use the query optimizer to identify the appropriate physical design selection from various proposed candidate designs [15]. ECOS also performs automated physical design, but at the different level, i.e., at the storage manager level. It does not rely on a query optimizer. Furthermore, ECOS is designed with the motivation of exploring new architectures for developing self-tuning DBMS instead of developing techniques to self-tune existing ones.

6 Conclusion and Future Work

In this paper, we presented ECOS, a customizable and online self-tuning storage manager. ECOS and evolution paths enable and use the fine-grained customization of storage structures at the table-level and column-level. In addition, ECOS and evolution paths allow storage structures to autonomically evolve (to more suitable storage structures) with the change in the data management needs, to maintain the desirable performance while keeping the human intervention at a minimum. We also presented a detailed evaluation and discussion of ECOS and evaluation paths showing the performance improvement and reduced resource consumption. As future work, we plan to enhance the presented dictionary based DSM schemes for better performance. ECOS self-tuning design makes it a suitable candidate for emerging cloud computing platforms for data services. We also intend to investigate the efficient utilization of multi-core and many-core parallel processors using the presented evolution mechanism. Once query processing is implemented, we want to integrate the presented evolution mechanism with query processing, and then we will be able to evaluate the ECOS using the full TPC-H benchmark. Transaction management is also an implementation specific future work for our ECOS prototype.

Acknowledgments. Syed Saif ur Rahman is a HEC-DAAD Scholar funded by Higher Education Commission of Pakistan and NESCOM, Pakistan.

References

1. Abadi, D.J., Madden, S.R., Ferreira, M.C.: Integrating compression and execution in column-oriented database systems. In: SIGMOD, pp. 671–682 (2006)
2. Abadi, D.J., Madden, S.R., Hachem, N.: Column-stores vs. row-stores: how different are they really? In: VLDB, pp. 967–980 (2008)
3. Agrawal, S., Chu, E., Narasayya, V.: Automatic physical design tuning: workload as a sequence. In: SIGMOD, pp. 683–694 (2006)
4. Batory, D.S.: On searching transposed files. *ACM Trans. Database Syst.* 4(4), 531–544 (1979)
5. Bender, M.A., Demaine, E.D., Farach-Colton, M.: Cache-oblivious B-trees. In: FOCS, pp. 399–409 (2000)
6. Chaudhuri, S., Weikum, G.: Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. In: VLDB, pp. 1–10 (2000)
7. Chaudhuri, S., Weikum, G.: Foundations of automated database tuning. In: SIGMOD, pp. 964–965 (2005)
8. Chen, S., Gibbons, P.B., Mowry, T.C., Valentin, G.: Fractal prefetching B+-Trees: optimizing both cache and disk performance. In: SIGMOD, pp. 157–168 (2002)
9. Chmiel, J., Morzy, T., Wrembel, R.: HOBI: Hierarchically Organized Bitmap Index for Indexing Dimensional Data. In: Pedersen, T.B., Mohania, M.K., Tjoa, A.M. (eds.) DaWaK 2009. LNCS, vol. 5691, pp. 87–98. Springer, Heidelberg (2009)
10. Copeland, G.P., Khoshafian, S.N.: A decomposition storage model. *SIGMOD Rec.* 14, 268–279 (1985)
11. Hellerstein, J.L.: Automated tuning systems: Beyond decision support. In: CMG, Computer Measurement Group, pp. 263–270 (1997)
12. Holloway, A.L., DeWitt, D.J.: Read-optimized databases, in depth. *Proc. VLDB Endow.* 1, 502–513 (2008)
13. Malik, T., Wang, X., Burns, R., Dash, D., Ailamaki, A.: Automated physical design in database caches. In: ICDE Workshop, pp. 27–34 (2008)
14. Morzy, M., Morzy, T., Nanopoulos, A., Manolopoulos, Y.: Hierarchical Bitmap Index: An Efficient and Scalable Indexing Technique for Set-Valued Attributes. In: Kalinichenko, L.A., Manthey, R., Thalheim, B., Wloka, U. (eds.) ADBIS 2003. LNCS, vol. 2798, pp. 236–252. Springer, Heidelberg (2003)
15. Papadomanolakis, S., Dash, D., Ailamaki, A.: Efficient use of the query optimizer for automated physical design. In: VLDB, pp. 1093–1104 (2007)
16. Patterson, D.A., Ditzel, D.R.: The case for the reduced instruction set computer. *SIGARCH Comput. Archit. News* 8, 25–33 (1980)
17. TPC-H, <http://www.tpc.org/tpch/>
18. ur Rahman, S.S.: Using Evolving Storage Structures for Data Storage. In: FIT, pp. 30:1–30:6 (2010)
19. ur Rahman, S.S., Schallehn, E., Saake, G.: ECOS: Evolutionary Column-Oriented Storage. Tech. Rep. FIN-03-2011, Department of Technical and Business Information Systems, Faculty of Computer Science, University of Magdeburg (2011)
20. Valduriez, P., Khoshafian, S.N., Copeland, G.P.: Implementation Techniques of Complex Objects. *VLDB*, 101–110 (1986)
21. Valgrind, <http://www.valgrind.org>
22. Weikum, G., Moenkeberg, A., Hasse, C., Zaback, P.: Self-tuning database technology and information services: from wishful thinking to viable engineering. In: VLDB, pp. 20–31 (2002)

X-HYBRIDJOIN for Near-Real-Time Data Warehousing

Muhammad Asif Naeem, Gillian Dobbie, and Gerald Weber

Department of Computer Science, The University of Auckland,
Private Bag 92019, Auckland, New Zealand
mnae006@aucklanduni.ac.nz,
{gill,gerald}@cs.auckland.ac.nz

Abstract. In order to make timely and effective decisions, businesses need the latest information from data warehouse repositories. To keep these repositories up-to-date with respect to end user updates, near-real-time data integration is required. An important phase in near-real-time data integration is data transformation where the stream of updates is joined with disk-based master data. The stream-based algorithm Mesh Join (MESHJOIN) has been proposed to amortize disk access over fast stream. MESHJOIN makes no assumptions about the data distribution. In real world applications, however, skewed distributions can be found, e.g, certain products are sold more frequently than the remainder of the products. The question arises, how much does MESHJOIN loose in terms of performance by not adapting to data skew. In this paper we perform a rigorous experimental study analyzing the possible performance improvements while considering typical data distributions. For this purpose we design an algorithm Extended Hybrid Join (X-HYBRIDJOIN) that is complementary to MESHJOIN in that it can adapt to data skew and stores parts of the master data in memory permanently, reducing the disk access overhead significantly. We compare the performance of X-HYBRIDJOIN against the performance of MESHJOIN. We take several precautions to make sure the comparison is adequate and focuses on the utilization of data skew. The experiments show that considering data skew offers substantial room for performance gains that cannot be used by non-adaptive approaches such as MESHJOIN.

Keywords: Near-real-time data warehousing, stream-based join, data transformation, performance and tuning.

1 Introduction

Near-real-time data warehouse deployments are driving an evolution to more aggressive data freshness levels. The tools and techniques for delivering these new service levels are evolving rapidly [1] [2]. In the beginning, most data warehouses refreshed all content fully during each load cycle. However, due to an increasing demand for information freshness, it became infeasible to meet business needs. Therefore the data acquisition mechanism in warehouses was changed from full

refreshment to an incremental refresh strategy, in which new data is added to the warehouse without requiring a complete reload [3] [4]. Although this strategy is more efficient than the traditional one, it is still batch-oriented as a fraction of the data is propagated towards the warehouse after a particular timestamp. In order to overcome update delays, these batch-oriented and incremental refresh strategies are being replaced with a continuous refresh strategy [5] [6]; that is, sales data are being captured and propagated to the data warehouse in near-real-time fashion in order to support high levels of data freshness.

An important operation in data integration is the transformation of the source data to a required format. Content enrichment is an example of such a transformation. In content enrichment master data attributes are added to source data [7]. A related example, or a special case of content enrichment, is the replacement of a *source data key* with a *warehouse key*. We consider an example of an inventory sales system, as shown in Figure 1. The data source is in this example shown as a *source table* and it contains attributes *product_id*, *quantity* and *date* with the *primary key* on attribute *product_id*. The *look-up table* that stores master data contains attributes *product_id*, *surrogate_key*, *product_name*, *sale_price* and *supplier_id* with an index on attribute *product_id*. The attribute *surrogate_key* is the primary key for the data warehouse. Let's assume, before loading the source table records into the data warehouse, they need to be enriched with certain information from the look-up table. In our example, the enriched attributes are *supplier_id* and *total*, and the *surrogate_key* has to be added as well. A join operator is required to perform these enrichment and key adding tasks. In the context of near-real-time data warehousing one of the significant factors for choosing the join operator is that both the inputs for the join come from different sources and arrive at different rates. The source data is not a table but a high volume stream and it has a bursty nature while the *look-up table* is disk-based. The access rate of the look-up table is comparatively slow due to disk I/O cost; therefore a bottleneck is created during the join execution. The challenge in this case is to eliminate this bottleneck by amortizing disk I/O cost over a fast stream of updates. An alternative approach would be to try to put the whole disk-based relation into memory. In some cases this alternative can be feasible. But still there are a number of scenarios where this alternative is not applicable e.g. if the join is to be performed on a single computer where the bulk of memory is used for other purposes. Similarly, for intermittent streams, a main memory approach would keep the memory occupied even when no stream data is incoming. In the limited-memory approaches here, in contrast there is no such waste of resources.

A novel algorithm Mesh Join (MESHJOIN) [8] [9] has been designed especially for joining a continuous stream with a disk-based relation, such as the scenario in active data warehouses. The algorithm makes no assumptions about data distribution or the organization of the master data. Experiments by the MESHJOIN authors have shown that the algorithm performs worse with skewed data. The MESHJOIN algorithm is a hash join, where the stream serves as the build input and the disk-based relation serves as the probe input. The algorithm performs

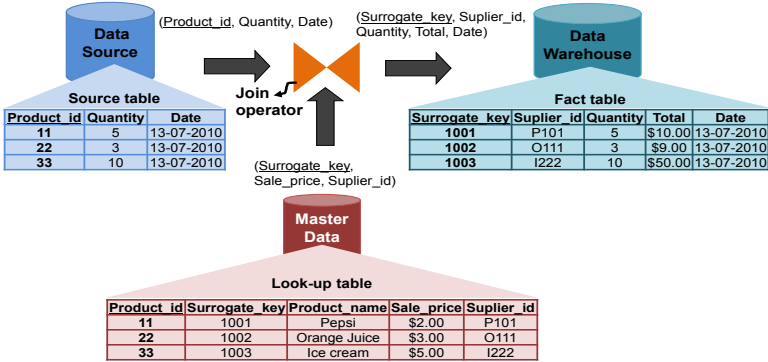


Fig. 1. An example of stream-based join

a staggered execution of the hash table build in order to load in stream tuples more steadily. However there are some issues such as suboptimal distribution of memory among the join components and an inefficient strategy for accessing the disk-based relation [12].

Although the MESHJOIN algorithm efficiently amortizes the disk I/O cost over fast input streams, the question remains how much potential for improvement remains untapped due to the algorithm not being able to adapt to common characteristics of real-world applications. In this paper we focus on one of the most common characteristics, a skewed distribution. Such distributions arise in practice, for example current economic models show that in many markets a select few products are bought with higher frequency [11]. Therefore, in the input stream, the sales transactions related to those products are the most frequent. In MESHJOIN, the algorithm does not consider the frequency of stream tuples, and does not need an index structure on the master data. This can be useful in some circumstances, but still in many other cases one obviously wants to use an index to gain maximum performance. We propose an adaptive algorithm called Extended Hybrid Join (X-HYBRIDJOIN). The key feature of X-HYBRIDJOIN is that the algorithm stores the most used portion of the disk-based relation, which matches the frequent items in the stream, in memory. As a result, this reduces the I/O cost substantially, which improves the performance of the algorithm.

X-HYBRIDJOIN has two major modifications compared with MESHJOIN. Firstly, the hash join component of X-HYBRIDJOIN is modified so that it can make use of an index. Secondly, X-HYBRIDJOIN caches frequently used master data. Since we want to compare MESHJOIN and X-HYBRIDJOIN, it is important to clarify, which change leads to the performance improvement. Therefore we also present an intermediate step, HYBRIDJOIN, which implements only the first modification, and we compare all three algorithms. Since our purpose is primarily to gauge the potential of skewed distributions, we consider a very clean, artificial dataset that exactly exhibits a well-understood type of skew, a power law.

The remainder of the paper is structured as follows. A review of the related work is presented in Section 2. In section 3 we describe the intermediate algorithm HYBRIDJOIN that already uses an index in the join process. In Section 4, we present the difference between HYBRIDJOIN and X-HYBRIDJOIN, and also derive the cost model for X-HYBRIDJOIN. The experimental study is discussed in Section 5 and finally Section 6 concludes the paper.

2 Related Work

In this section, we present an overview of the previous work that has been done in this area, focusing on that which is closely related to our problem domain.

A novel algorithm Mesh Join (MESHJOIN) [8] [9] has been designed especially for joining a continuous stream with a disk-based relation, such as the scenario in active data warehouses. Although it is an adaptive approach, there are some research issues such as suboptimal distribution of memory among the join components and an inefficient strategy for accessing the disk-based relation. The algorithm makes no assumptions about data distribution or the organization of the master data. However, the problem that we are addressing is that MESHJOIN’s performance is directly coupled to the size of the master data table, and its performance is inversely proportional to the size of the master data table. This is an undesired behavior if the master data becomes very large, and our analysis will show that it is indeed an unnecessary behavior. The problem becomes obvious if we consider that the master data table contains a large part that is never joined with the stream data.

A revised version of MESHJOIN called R-MESHJOIN (reduced Mesh Join) [12] has been presented by us. It addresses the issue of optimal distribution of memory among the join components. In this algorithm a new strategy for memory distribution among the join components is introduced capturing real constraints. However the issue of an inefficient strategy for accessing the disk-based relation still exists in R-MESHJOIN.

One approach for improving MESHJOIN has been a partition-based join algorithm [10] which can also deal with stream intermittence. It uses a two-level hash table in order to attempt to join stream tuples as soon as they arrive, and uses a partition-based waiting area for other stream tuples. For the algorithm in [10], however, the time that a tuple waits for execution is not bounded. We are, however, interested in a join approach where the time in which a stream tuple is joined is guaranteed.

3 Index-Based Hash Join Architecture: HYBRIDJOIN

In this section we introduce the HYBRIDJOIN algorithm, which implements our first modification of MESHJOIN in order to make use of a non-clustered index. We introduce the join architecture for HYBRIDJOIN. This will be used, with a single modification, for the algorithm X-HYBRIDJOIN as well.

HYBRIDJOIN joins a disk-based relation R with a stream S . We assume a non-clustered index on R for the join attribute, and we assume that the join attribute is unique within the master data. This is a very natural set of assumptions and matches with common application domains, for example in key exchange applications. By only requiring a non-clustered index, we keep our assumptions as minimal as possible.

The memory architecture used in HYBRIDJOIN and in X-HYBRIDJOIN is shown in Figure 2. The main memory components are a disk buffer, a hash table, a queue and a stream buffer while the disk-based relation R and stream S are the inputs. In our algorithm, we assume that R has an index with the join attribute as the key.

The stream is used as the build input. This means that the algorithm keeps stream tuples in a hash table which occupies the largest share of the memory, and the hash table is filled with the next pending stream tuples to its full capacity. Additionally we keep identifiers of the stream tuples in a queue which allows random deletion, the simplest implementation is a doubly linked list.

HYBRIDJOIN is an iterative algorithm, and in each iteration it uses a partition of the disk-based relation R as a probe input. For that purpose, the partition is loaded into the disk buffer. In HYBRIDJOIN, the disk buffer contains only this partition, later in X-HYBRIDJOIN the partition will only occupy one part of the disk buffer. After that, the algorithm performs the typical operation of a hash join, i.e., it loops over all the tuples of the disk buffer and looks them up in the hash table. In the case of a match, the algorithm generates that stream tuple as an output.

Also, in each iteration, HYBRIDJOIN evicts stream tuples that have been matched. This is justified through the assumption that the join attribute is unique in R . Evicting a tuple means it is deleted from the hash table and the queue. The algorithm also keeps a counter w of the evicted tuples. After processing the whole disk buffer, the algorithm reads w new tuples from the stream buffer, loads them in the hash table along with entering their identifiers in the queue.

For choosing the next partition of R , HYBRIDJOIN looks at the join attribute of the oldest stream tuple in the queue. Using the index, it loads the partition of R with that join attribute value into the disk buffer. It is this last step which makes HYBRIDJOIN adaptive, because in HYBRIDJOIN, every loaded partition matches at least one stream tuple. As a simple example, consider R has a section that is not referred to in the stream, for example an obsolete group of products. In MESHJOIN, this section would still be loaded, while in HYBRIDJOIN it would not be loaded, because no stream tuple will trigger the loading of that section.

HYBRIDJOIN works for any data distribution, as MESHJOIN does. However, in practice, certain distributions are common. Current research has shown that sales data typically follows a power law, or Zipfian distribution [11]. The power law is characterized by its exponent. For an exponent <1 the distribution is said to have a long tail, for an exponent >1 the distribution has a short

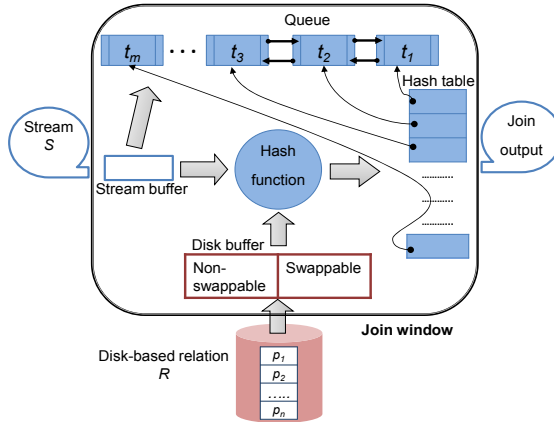


Fig. 2. Architecture of HYBRIDJOIN and X-HYBRIDJOIN. The only difference between the two algorithms is that in X-HYBRIDJOIN the disk buffer is split and its two parts are treated differently, as explained in the text.

tail. For exponent 1 we get the distribution of Zipf’s law, which gave rise to the general term Zipfian distribution. In sales, the 80/20 rule is used to model the scenario where the frequency of selling a small number of products is significantly higher compared to the rest of the product, often simplified in the 80/20 rule [13]. The 80/20 rule of thumb has commonly been observed in commercial applications [13] [14]. The 80/20 rule corresponds to an exponent slightly smaller than 1 [13].

Our aim is to describe an algorithm that takes advantage of the likely distribution of the data. Therefore we created a dataset generator that can create artificial data sets following a power law with an exponent that can be chosen freely. In all our experiments, the master data is sorted with respect to the access frequency.

4 X-HYBRIDJOIN

In this section, we describe our second algorithm, X-HYBRIDJOIN which is an extension of HYBRIDJOIN.

4.1 Difference between X-HYBRIDJOIN and HYBRIDJOIN

As we will see later, the service rate (number of tuples processed per second) of HYBRIDJOIN increases as the exponent of the distribution goes above 1 i.e. as the distribution gets closer to a short-tailed distribution. However, if a distribution is fairly short-tailed, then many matches are with the most frequent tuples. So the question arises, how much can be gained in terms of performance by buffering the most frequent tuples permanently, and this gives rise to X-HYBRIDJOIN.

Algorithm 1. Pseudo-code for X-HYBRIDJOIN

Input: A disk-based relation R with an index on join attribute and a stream of updates S **Output:** $S \bowtie R$ **Parameters:** w tuples of S and a partition p_i of R **Method:**

```

1: LOAD first partition  $p_1$  of  $R$  into the non-swappable part of the disk buffer.
2:  $w \leftarrow h_S$ 
3: while (true) do
4:   if (available stream tuples  $\geq w$ ) then
5:     READ  $w$  tuples from the stream buffer, load them into  $H$  and enqueue their
     join attribute values into  $Q$ .
6:      $w \leftarrow 0$ 
7:   end if
8:   for each tuple  $r$  in  $p_1$  do
9:     if  $r \in H$  then
10:      OUTPUT  $r \bowtie H$ 
11:       $w \leftarrow w + \text{number of matching tuples found in } H$ 
12:      DELETE all matched tuples from  $H$  and the corresponding nodes from  $Q$ .
13:    end if
14:  end for
15:  READ the oldest join attribute value from  $Q$ .
16:  LOAD a disk partition  $p_i$  (where  $2 \leq i \leq n$ ) of  $R$  into the swappable part of the
  disk buffer using the join attribute value as an index.
17:  for each tuple  $r$  in  $p_i$  do
18:    if  $r \in H$  then
19:      OUTPUT  $r \bowtie H$ 
20:       $w \leftarrow w + \text{number of matching tuples found in } H$ 
21:      DELETE all matched tuples from  $H$  and the corresponding nodes from  $Q$ .
22:    end if
23:  end for
24: end while

```

The difference between the two algorithms is that in X-HYBRIDJOIN we divide the disk buffer into two parts. One part stores the most popular pages of disk-based relation R in memory permanently, and we call this the non-swappable part of the disk buffer. The other part of the disk buffer is swappable and is used to load partitions from the remainder of relation R into memory in the same way as in the HYBRIDJOIN algorithm. As a natural default setting, we assign the same amount of memory to both parts.

4.2 Algorithm

Once the available memory is distributed among the join components, the algorithm is ready to execute according to the procedure described in Algorithm 1. Before starting the actual join execution, the algorithm reads a particular portion of the disk-based relation R into the non-swappable part of the disk buffer

(line 1). In the beginning all slots in the hash table H are empty; therefore, h_S is assigned to w (line 2). In the abstract level description, the algorithm contains two kind of loops. One is called the outer loop, which is an endless loop (line 3). The key objective of the outer loop is to build the stream in the hash table. Within the outer loop, the algorithm runs two independent inner loops. One loop implements the probing module for the non-swappable part of the disk buffer, while the other inner loop implements the probing of the swappable part of the disk buffer. As the outer loop begins, the algorithm observes the status of the stream buffer. If stream is available the algorithm reads the w tuples from the stream buffer and loads them into the hash table, while also enqueueing their attribute values into the queue. After completing the stream input the algorithm resets w to 0 (lines 4 to 7). The algorithm then executes the first inner loop, in which it reads all tuples one-by-one from the non-swappable part of the disk buffer and looks them up into the hash table. In the case of a match, the algorithm generates the join output. Due to the multi-hash-map, there can be more than one match against one disk tuple. After generating the join output the algorithm deletes all matched tuples from the hash table, along with the corresponding nodes from the queue. The algorithm also increments w with the number of vacated slots in the hash table (lines 8 to 14). Before starting the second inner loop, the algorithm reads the oldest value of the join attribute from the queue (line 3) and loads a disk partition p_i (where $2 \leq i \leq n$) into the swappable part of the disk buffer, using that join attribute value as an index (lines 15 and 16). As the specified disk partition is loaded into the swappable part of the disk buffer, the algorithm starts the second inner loop and repeats all the steps described in the first inner loop (lines 17 to 23).

Note: If we switch-off the first inner loop (lines 8 to 14), the algorithm works as HYBRIDJOIN.

4.3 Cost Model

In this section we derive the general formulae for calculating the cost for our proposed X-HYBRIDJOIN. We derive equations for memory and processing time of X-HYBRIDJOIN. Equation 1 describes the total memory used to implement the algorithm except for the stream buffer; whereas Equation 2 calculates the processing cost for w tuples. The symbols used to measure the costs are specified in Table 1.

Memory cost. In X-HYBRIDJOIN, the disk buffer is divided into two equal parts. One is swappable, the other is non-swappable. As said before, the largest share of the total memory is used for the hash table; a much smaller portion is used for the disk buffer. The queue size is a constant fraction of the hash table size. The memory for each component of X-HYBRIDJOIN can be calculated as shown below.

Memory reserved for the swappable and non-swappable parts = $v_P + v_P = 2v_P$
(in the case of HYBRIDJOIN it is v_P only).

Memory for the hash table = $\alpha(M - 2v_P)$.

Table 1. Notations used in cost estimation of X-HYBRIDJOIN

Parameter name	Symbol
Total allocated memory (<i>bytes</i>)	M
Service rate (<i>processed tuples/sec</i>)	μ
Input size (=number of matching tuples in previous iteration)	w
Stream tuple size (<i>bytes</i>)	v_S
Size of each swappable and non-swappable part (<i>bytes</i>) (=size of 1 disk partition)	v_P
Size of disk tuple (<i>bytes</i>)	v_R
Size of each swappable and non-swappable part (<i>tuples</i>)	$d_T = \frac{v_P}{v_R}$
Memory weight for the hash table	α
Memory weight for the queue	$1-\alpha$
Cost to read one disk partition into the disk buffer (<i>nanosecs</i>)	$c_{I/O}(v_P)$
Cost to lookup one tuple in the hash table (<i>nanosecs</i>)	c_H
Cost to generate the output for one tuple (<i>nanosecs</i>)	c_O
Cost to remove one tuple from the hash table and the queue (<i>nanosecs</i>)	c_E
Cost to read one stream tuple into the stream buffer (<i>nanosecs</i>)	c_S
Cost to append one tuple into hash table and the queue (<i>nanosecs</i>)	c_A
Total cost for one loop iteration of X-HYBRIDJOIN (<i>secs</i>)	c_{loop}

Memory for the queue = $(1 - \alpha)(M - 2v_P)$.

The total memory used by X-HYBRIDJOIN can be determined by aggregating all of the above.

$$M = 2v_P + \alpha(M - 2v_P) + (1 - \alpha)(M - 2v_P) \quad (1)$$

Currently we do not include the memory reserved by the stream buffer because of its small size (0.05 MB has been sufficient in all our experiments).

Processing cost. In this section, we calculate the processing cost for the proposed X-HYBRIDJOIN. We denote the cost for one loop iteration of the algorithm as c_{loop} and express it as the sum of the costs for the individual operations. We first calculate the processing cost for each component separately.

Cost to read swappable or non-swappable parts of the disk buffer = $c_{I/O}(v_P)$.

Cost to look-up swappable and non-swappable parts of the disk buffer in the hash table = $d_T \cdot c_H + d_T \cdot c_H = 2d_T \cdot c_H$ (in the case of HYBRIDJOIN it is $d_T \cdot c_H$ only).

Cost to generate the output for w matching tuples = $w \cdot c_O$.

Cost to remove w tuples from the hash table and the queue = $w \cdot c_E$.

Cost to read w tuples from stream S into the stream buffer = $w \cdot c_S$.

Cost to append w tuples in the hash table and the queue = $w \cdot c_A$.

As the non-swappable part of the disk buffer is read only once before the execution starts we exclude it. By aggregating the terms, the total cost for one loop iteration is:

$$c_{loop}(secs) = 10^{-9}[c_{I/O}(v_P) + 2d_T \cdot c_H + w(c_O + c_E + c_S + c_A)] \quad (2)$$

Table 2. Data specification

Parameter	value
Disk-based data	
Size of disk-based relation R	0.5 million to 8 million tuples
Size of each tuple	120 bytes
Stream data	
Size of each tuple	20 bytes
Size of each node in queue	12 bytes
Benchmark	
Based on	Zipf’s law
Characteristics	Bursty and self-similar

For all c_{loop} seconds the algorithm processes w tuples of stream S ; therefore, the service rate μ can be calculated by dividing w by the cost for one loop iteration as shown in Equation 3.

$$\mu = \frac{w}{c_{loop}} \quad (3)$$

5 Experiments

We performed experiments to compare the performance of our algorithms with MESHJOIN. We also validate the measured cost by comparing it with the calculated cost for each algorithm. As mentioned before, we use synthetic data sets with a known skew.

5.1 Experimental Setup

Hardware Specifications: We carried out our experiments on a Pentium-IV 2X2.13GHz machine under WindowsXP. The maximum memory we allocated for our experiments is 250MB. We implemented the algorithm in Java. To measure the memory and processing time, we used built-in plugins provided by Apache and Java API respectively.

Data specifications: The synthetic workload that we used to test the algorithms was generated using Zipf’s Law with exponent 1. The generated stream has two additional characteristics known as burstyness and self similarity. The detailed specifications of the data set that we used for analysis are shown in Table 2. The relation R is stored on disk using MySQL 5.0 databases. To measure the cost for each I/O operation accurately we set the fetch size for the *ResultSet* equal to the size of one partition on disk. X-HYBRIDJOIN needs to store multiple values in the hash table against one key value. However, the hash table provided by the standard Java API does not support this feature;

therefore, we have used the Multi-Hash-Map from Apache as the hash table in our experiments.

Measurement strategy: We define the performance of the algorithms as service rate, with a higher service rate being better. The service rate has been measured by calculating the number of tuples processed in a unit second. In each experiment, the algorithm is executed for one hour. We started our measurements after 20 minutes and keep measuring for 20 minutes. For added accuracy, we took three readings for each specification and then calculated the average. Where required we also calculated the confidence interval by considering 95% accuracy. The calculation of confidence interval is based on 4000 measurements for one setting. Moreover, during the execution of the algorithm no other application was running in parallel.

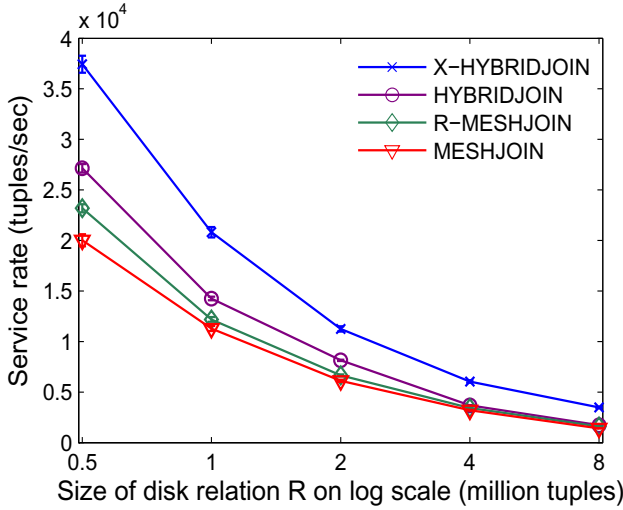
5.2 Experimental Results

In our experimental study, we analyzed the results from three different perspectives. Firstly, we compare the performance of both HYBRIDJOIN and X-HYBRIDJOIN with the other related algorithms. Secondly, we examine the role of the non-swappable part of the disk buffer in stream processing. Finally, we validate our predicted cost model through experiment.

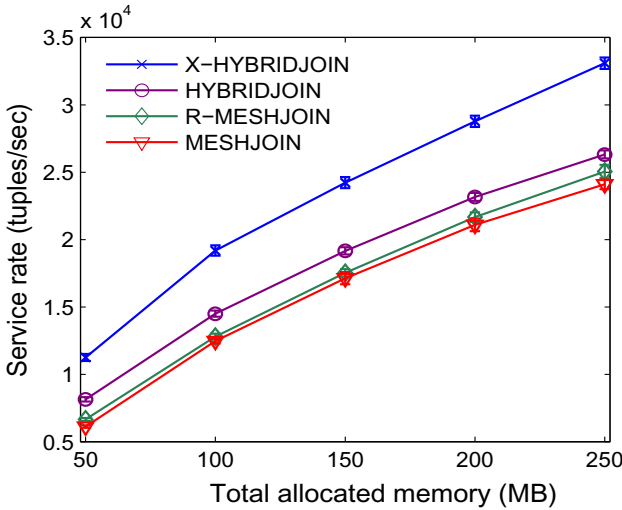
Performance comparisons: The two possible parameters that can vary and directly affect the performance of the algorithms under test are the total available memory for the algorithm and the size of the disk-based relation. In our experiments, we tested the algorithms for different values of these parameters and compared their performance.

Performance comparisons for varying size of the disk-based relation: In the experiment shown in Figure 3(a), we assumed the total allocated memory for the join was fixed while the size of the disk-based relation R was grown exponentially. Figure 3(a) shows that for all sizes of R performance of X-HYBRIDJOIN is substantially better than all the other approaches. Another key observation from the figure is that when R is 0.5 million the performance of HYBRIDJOIN is almost 70% of X-HYBRIDJOIN and when R is equal to 8 million this percentage decreases to 50%. This means that the performance of the other algorithms decreases more sharply compared to X-HYBRIDJOIN when R increases.

Performance comparisons when the size of available memory varies: In our second experiment, we analysed the performance of X-HYBRIDJOIN using different memory budgets while the size of R is fixed (2 million tuples). Figure 3(b) presents the results of our experiment. The figure indicates that, for all memory budgets, the performance of X-HYBRIDJOIN is again significantly better than all the other algorithms. The reason behind this improvement is our intuition about X-HYBRIDJOIN. In our calculations, introducing the non-swappable part in X-HYBRIDJOIN can save about 33% of the disk I/O cost. Although keeping the non-swappable part in memory increases the look-up cost



(a) Performance comparison with 95% confidence interval while $M = 50MB$ and R varies



(b) Performance comparison with 95% confidence interval while $R = 2$ million tuples and M varies

Fig. 3. Performance comparisons

and reduces the memory for the hash table, both these factors are very small compared to the disk I/O cost.

From the experiments we can see that HYBRIDJOIN performs consistently slightly better than MESHJOIN and R-MESHJOIN. However, the improvement is rather modest. Our experiments show that the main performance gain of

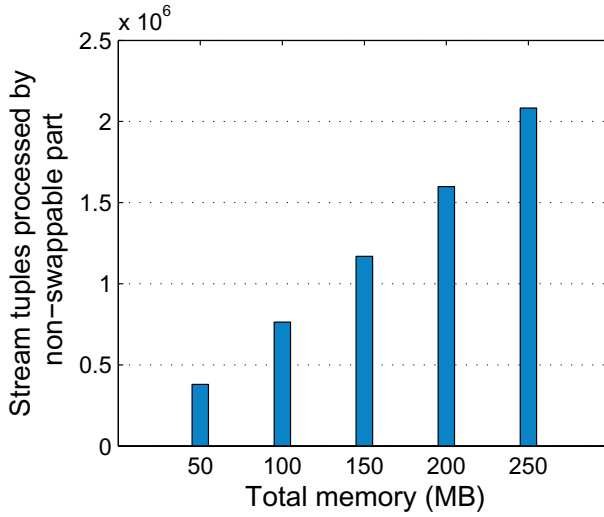


Fig. 4. Total number of stream tuples processed with non-swappable part of disk buffer in 4000 iteration

X-HYBRIDJOIN is due to the second improvement, the introduction of a non-swappable part in the disk-buffer.

Role of the non-swappable part in stream processing. To get a better understanding of the role of the non-swappable part of the disk buffer, we performed

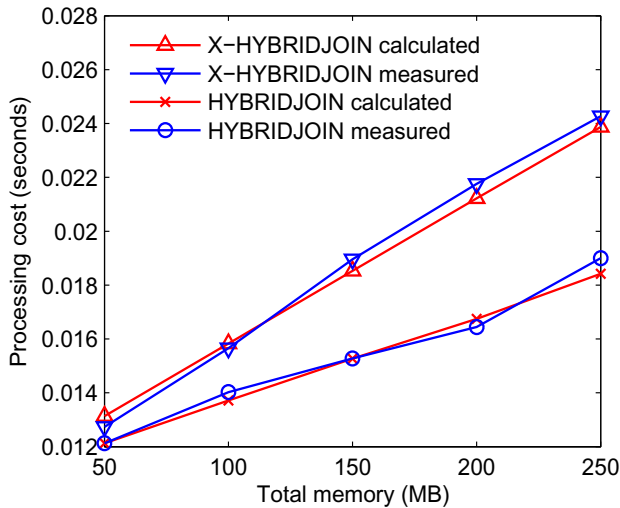


Fig. 5. Cost validation

an experiment where we counted the stream tuples which are processed using only the non-swappable part of disk buffer. The results of this experiment are shown in Figure 4. As before, we set the size of the non-swappable part to be equal to the size of the swappable part. It is clear from the figure that in 4000 iterations when the memory budget is 50 MB and the size of R is 2 million tuples, about 0.4 million stream tuples are processed through the non-swappable part of the disk buffer and this number increases if we increase the total allocated memory. For 250 MB memory with the same size of R (2 million tuples), this amount reaches more than 2 million. In the other algorithms, since this non-swappable part is loaded from the disk each time, the I/O cost increases significantly.

Cost validation. We validate our results by comparing the predicted cost with the measured cost. Figure 5 presents the comparisons of both costs for each algorithm. In the figure, it can be seen that for each algorithm the predicted cost closely matches the measured cost, which is evidence of the consistency of our study.

6 Conclusions and Future Work

In this paper, we explored the potential improvement for stream-based joins if characteristics of the data such as skew are taken into account. MESHJOIN performs worse with skewed distributions, which is a problem since these distributions are common in real world applications. We presented a robust algorithm called X-HYBRIDJOIN (Extended Hybrid Join) with two major modifications over MESHJOIN. The first modification is the use of an index on disk-based master data. The second modification is that X-HYBRIDJOIN caches the most frequent tuples of master data. As a result it reduces the disk access and improves the performance substantially. To validate our arguments we implemented the prototypes for both modifications and carried out experiments comparing the different algorithms. We provided open source implementations of our algorithms.

In the future we plan to tune the X-HYBRIDJOIN algorithm in order to utilize the available memory resources optimally.

Source URL: The source of our implementations and pseudo-codes can be downloaded using the given URL:

<https://www.cs.auckland.ac.nz/research/groups/serg/src/>

References

1. Karakasidis, A., Vassiliadis, P., Pitoura, E.: ETL queues for active data warehousing. In: IQIS 2005: Proceedings of the 2nd International Workshop on Information Quality in Information Systems, pp. 28–39. ACM, New York (2005)
2. Naem, M.A., Dobbie, G., Weber, G.: An Event-Based Near Real-Time Data Integration Architecture. In: Enterprise Distributed Object Computing Conference Workshops, pp. 401–404. IEEE, Munich (2008)

3. Labio, W., Yang, J., Cui, Y., Garcia-Molina, H., Widom, J.: Performance Issues in Incremental Warehouse Maintenance. In: VLDB 2000: Proceedings of the 26th International Conference on Very Large Data Bases, San Francisco, CA, USA, pp. 461–472 (2000)
4. Labio, W.J., Wiener, J.L., Garcia-Molina, H., Gorelik, V.: Efficient resumption of interrupted warehouse loads. *SIGMOD Rec.* 29(2), 46–57 (2000)
5. Nguyen, A., Tjoa, A.: Zero-Latency data warehousing for heterogeneous data sources and continuous data streams. In: iiWAS 2003 - The Fifth International Conference on Information Integration and Web-based Applications Services, pp. 55–64. Austrian Computer Society, OCG (2003)
6. Golab, L., Johnson, T., Seidel, J.S., Shkapenyuk, V.: Stream warehousing with DataDepot. In: Proceedings of the 35th SIGMOD International Conference on Management of Data, Providence, Rhode Island, USA, pp. 847–854 (2009)
7. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Longman Publishing Co., Boston (2003)
8. Polyzotis, N., Skiadopoulos, S., Vassiliadis, P., Simitsis, A., Frantzell, N.E.: Supporting Streaming Updates in an Active Data Warehouse. In: IEEE 23rd International Conference on Data Engineering, ICDE 2007, Istanbul, Turkey, pp. 476–485 (2007)
9. Polyzotis, N., Skiadopoulos, S., Vassiliadis, P., Simitsis, A., Frantzell, N.: Meshing Streaming Updates with Persistent Data in an Active Data Warehouse. *IEEE Trans. on Knowl. and Data Eng.* 20(7), 976–991 (2008)
10. Chakraborty, A., Singh, A.: A partition-based approach to support streaming updates over persistent data in an active datawarehouse. In: IPDPS 2009: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing, pp. 1–11. IEEE Computer Society, Washington, DC, USA (2009)
11. Anderson, C.: The Long Tail: Why the Future of Business is Selling Less of More (2006), Hyperion
12. Naem, M.A., Dobbie, G., Weber, G.: R-MESHJOIN for Near-real-time Data Warehousing. In: DOLAP 2010: Proceedings of the ACM 13th International Workshop on Data Warehousing and OLAP. ACM, Toronto (2010)
13. Knuth, D.E.: The art of computer programming, pp. 400–401. Addison-Wiley, Reading, Mass (1968)
14. Heising, W.P.: Note on random addressing techniques. *IBM Systems Journal* 2(2), 976–991, 114–115 (1963)

Achieving High Freshness and Optimal Throughput in CPU-Limited Execution of Multi-join Continuous Queries[★]

Abhishek Mukherji, Elke A. Rundensteiner, and Matthew O. Ward

Computer Science Department, Worcester Polytechnic Institute, Worcester MA, USA
{mukherab,rundenst,matt}@wpi.edu

Abstract. Due to high data volumes and unpredictable arrival rates, continuous query systems processing expensive queries in *real-time* may fail to keep up with the input data streams - resulting in buffer overflow and uncontrolled data loss. We explore *join direction adaptation* (JDA) to tackle CPU-limited processing of multi-join stream queries. The existing JDA solutions allocate the scarce CPU resources to the most productive half-way join within a *single* operator. We instead leverage the operator *interdependencies* to optimize the overall query throughput. We identify *result staleness*, typically ignored by most state-of-the-art techniques, as a critical issue in CPU-limited processing. It gets further aggravated if throughput optimizing techniques are employed. We establish the novel *path-productivity* model and the *Freshness* predicate. Our proposed *JAQPOT* approach is the first integrated solution to achieve *near optimal* query throughput while also guaranteeing freshness satisfiability. JAQPOT runs in quadratic time of the number of streams irrespective of the query plan shape. Our experimental study demonstrates the superiority of JAQPOT in achieving higher throughput than the state-of-the-art JDA strategy while also fulfilling freshness predicates.

1 Introduction

Motivation. *Data Stream Management Systems* (DSMS) [2, 3, 15] are in high demand for real-time decision support as they transform huge amounts of streaming data into *usable* knowledge. Due to rapid expansions in the diversity of data sources and the volume of data these sources deliver, DSMS are faced with the challenge of processing user queries demanding *real-time* responsiveness even under conditions of unpredictability, high and bursty data volumes.

Windowed joins across streams, while among the most common queries in DSMS applications, are more costly compared to other operations such as selection, aggregation and projection [8, 9, 11]. When processing complex join queries, either the processor may fail to keep up with the arrival rates of the streams (the CPU-limited case) or the available main memory may become insufficient to hold all relevant tuples (the memory-limited case). For queries composed of joins with large states across multiple high-speed data streams, the system is even more prone to such resource deficiencies.

[★] This work was supported by NSF grants IS-0812027, CCF-0811510, IIS-0917017 and IIS-1018443.

```
SELECT B.symbol, B.price
FROM stocksNYC A [WIN 10 mins], stocksTokyo B [WIN 10 mins]
WHERE A.symbol = "GOOG" AND B.Volume > A.Volume
```

Fig. 1. Example Query

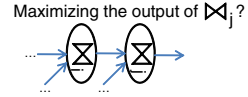


Fig. 2. A pipeline of join operators

Gedik et al. [8] observe that with increasing stream arrival rates and large join states, the CPU typically becomes strained before the memory does. Temporary data flushing [11] and compressed data representations further counteract the chances of a memory-limited scenario. If under duress complete results can no longer be produced at run-time, then the DSMS must employ the available resources to ensure the production of maximal run-time throughput (output rate). Therefore, in this work, we aim at optimizing the throughput of *multi-join queries* in CPU-limited cases.

When resources are limited, yet another pressing issue, namely, *result staleness* arises. In Query Q1 (Fig. 1) a stock trader is interested in *the companies whose stocks got traded at Tokyo in higher volumes than Google stocks traded in NYC*. He wants the comparable transactions to happen *within 10 minutes of each other*. For real-time decision making, the DSMS may be required to produce results continuously (say, once every minute). However, if the system faces high workloads and backlogs in processing, result tuples may get delayed. For example, the trader may receive results about transactions that took place 15 minutes before the current time. Such results, despite satisfying the 10-minute window predicate, would be considered *stale* and useless by the trader. Clearly, *high throughput results with no freshness guarantees are unacceptable in real-time applications as they may be producing results already deemed useless*.

In addition to the *WINDOW* predicate, the trader may want to specify a *freshness* predicate to indicate his *tolerance to staleness*. A *freshness* predicate may be defined on each stream, i.e., 12 mins for *stocksNYC* whereas 15 mins for *stocksTokyo*. To the best of our knowledge, our work is the first to identify the *result staleness* problem in the context of resource-limited execution of multi-join plans and tackles the dual problems of achieving optimal throughput while satisfying freshness of the join results.

The State-of-the-art. Two directions for tackling join queries under *computing* limitations are *load shedding* [4, 9, 16] and *join direction adaptation* (JDA) [8, 10]. The main focus of load shedding is to reduce the input rates by directly dropping tuples from the source streams [4]. This makes the plan incapable of recuperating with the production of accurate results in moments of low workloads as data is permanently lost.

Unlike load shedding, JDA preserves in-memory tuples as per the join semantics for opportunities of joining with future incoming tuples. Existing JDA techniques [8, 10] exploit the asymmetry in the productivities of half-way join directions within a join operator. However, JDA techniques have so far been explored only in the context of a *single* join operator. We demonstrate in this work that new challenges arise in the multi-join case. A detailed review of the related work is provided in Sec. 6.

Research Challenges. In general, the ability of multi-join queries to achieve *high* result throughput and to maintain result freshness under heavy workloads relies on resolving the following aspects of the problem:

- While *operator scheduling* [6, 7] tends to allocate resources at the *coarse* granularity of query operators, **adaptation at a finer granularity within the operators** is required to produce optimal throughput.
- The existing JDA technique [10] optimizes every join operator individually. In a pipeline of join operators (Fig. 2), an uncoordinated attempt to optimize operators \bowtie_i and \bowtie_j individually may jeopardize the real goal of optimizing the overall query throughput. The join operators within a multi-join plan are **interdependent**, namely, an operator depends on the output of its upstream¹ operator(s) for input. Consideration of **operator interdependency is crucial** for a successful plan-level join direction adaptation.
- We identify **result staleness**² as a critical issue for CPU-limited processing of multi-join queries. The biased resource allocation by **JDA** may potentially aggravate this problem.

Proposed Approach. Unlike load shedding that discards data once the system is on the verge of crashing from overload, we propose to preemptively allocate the available CPU resources with the goal to achieve maximal throughput. We design, develop and evaluate a **synchronized** join adaptation strategy at the plan level that tackles the **result staleness** problem while maximizing the overall throughput of the query. We summarize our contributions as follows:

1. We demonstrate that the state-of-the-art **JDA** [10] technique fails to achieve optimal throughput for the **CPU-limited** processing of *multi-join* plans (Sec. 3).
2. We establish the **path productivity** metric as the plan-level throughput contribution of each input stream (Sec. 4.1).
3. We formulate the query throughput maximization as a **knapsack** problem and propose a **Greedy Path Productivity-based Adaptation (GrePP)** to solve it (Sec. 4.2).
4. We identify result staleness as a key challenge under CPU-limited scenarios and formulate the **freshness satisfiability** as a **weighted set-cover** problem (Sec. 4.3).
5. We integrated the above two strategies into the JAQPOT algorithm (Sec. 4.4). To the best of our knowledge, this is the first solution that guarantees fulfillment of **result freshness** predicates while achieving **near optimal** query throughput. We further note that JAQPOT achieves this effective adaptation in **quadratic time in the number of input streams**.
6. Our experimental study (Sec. 5) demonstrates the superiority of JAQPOT over the state-of-the-art JDA solution in a large set of tested cases.

2 Preliminaries

2.1 Background

In this paper we focus on multi-join plans composed of sliding window binary join operators. We assume standard semantics as in CQL [3]. We use the **unit-time basis**

¹ Operators closer to the stream input are **upstream** and those closer to the query output are **downstream**.

² This challenge is not identified by prior work [4, 9, 16].

Symbol	Meaning
t^l	Tuple of stream I
$t^l.ts$	Timestamp of tuple t^l
λ_i	Arrival rate of stream I
$ S_i $	Window size of state I
σ_i	Selectivity of join on state S_i
λ'_i	Probe allowance of stream I, ($\leq \lambda_i$).

Fig. 3. List of notation

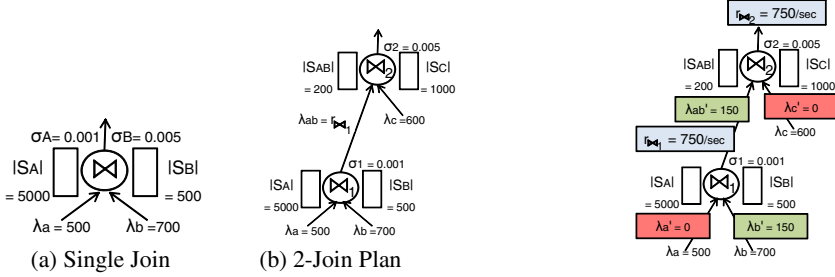


Fig. 4. Join plans with parameter settings

Fig. 5. JDA over a 2-join plan

cost model proposed by Kang et al. [10] that computes the join cost in terms of the three sub-tasks, namely, *probe*, *insert* and *purge* operations. For simplicity, the model assumes *count-based* windows. The key idea is that **the cost of probe dominates the total join cost while insert and purge operations are relatively inexpensive**. For details on the model and its extension to *time-based* windows refer to [10]. Fig. 3 lists the notation.

Throughput. The run-time throughput (Eq. 1) of a join operator $A \bowtie B$ (Fig. 4.a) consists of two contributing half-way join components, namely, $r_{\bowtie} = (a \times S_B)$ and $r_{\bowtie} = (b \times S_A)$. Throughput is also called the *output rate* and is defined as *the number of joined tuples produced per time unit*. For tuple t^A , S_A is the *own* state whereas S_B is the *partner* state.

$$r_{\bowtie} = r_{\bowtie} + r_{\bowtie} = \lambda_a \times \sigma_B \times |S_B| + \lambda_b \times \sigma_A \times |S_A| \quad (1)$$

CPU Limitation in a Join. When CPU is limited, the throughput of $A \bowtie B$ can be re-written as in Eq. 2. The total *available computing resources*, denoted as μ , may be determined from the system. Terms *available resources* and *service rate* are used interchangeably.

$$r_{\bowtie} = r_{\bowtie} + r_{\bowtie} = \lambda'_a \times \sigma_B \times |S_B| + \lambda'_b \times \sigma_A \times |S_A|, \quad \lambda'_a + \lambda'_b \leq \mu. \quad (2)$$

In Eq. 2, the μ resources allocated to a join is divided between the two half-way joins. Stream A is assigned a *probe allowance*, denoted by λ'_a , which is a portion μ_a of μ not exceeding the input rate λ_a , i.e., $\lambda'_a = \min(\mu_a, \lambda_a)$. Similarly, $\lambda'_b = \min((\mu - \mu_a), \lambda_b)$. As the *probe* cost dominates the total join cost, the resource restriction only affects

the *probe*. All input tuples undergo the *insert* and *purge* operations. In Fig. 4a A⋈B have input rates $\lambda_a = 500$ and $\lambda_b = 700$ tuples/sec. Assume $\mu = 300$ tuples/sec as the available resources. Therefore, a subset³ of 300 tuples out of the 1200 ($= 500 + 700$) tuples from either of the input streams is used for the *probe* operation. However, all 1200 arriving tuples undergo *insert* and *purge* operations every time unit. Thus, the μ resources (here 300 tuples/sec) must be divided among the *probe allowances* (here λ'_a and λ'_b) for throughput maximization.

2.2 Problem Definition

Now, we define our two target problems, namely, achieving *optimal throughput* and tackling *result staleness* in CPU-limited execution of *multi-join*.

CPU-limited Execution of Multi-Join Plans. In the 2-join plan of Fig. 4b⁴ composed of \bowtie_1 and \bowtie_2 , the *throughput optimization* problem is quite different from the single operator case. Now, the goal is to maximize the *throughput* $r_{\bowtie_{root}}$ of the root operator (here \bowtie_2).

$$\mathbf{r}_{\bowtie_1} = \lambda'_a \times \sigma_B \times |S_B| + \lambda'_b \times \sigma_A \times |S_A|; \mathbf{r}_{\bowtie_2} = \lambda'_{ab} \times \sigma_C \times |S_C| + \lambda'_c \times \sigma_{AB} \times |S_{AB}|, \quad (3)$$

$$\lambda'_a + \lambda'_b + \lambda'_{ab} + \lambda'_c \leq \mu.$$

Equation 3 depicts the CPU-limited case in a *multi-join* plan, where μ needs to be divided among four half-way joins, namely, λ'_a , λ'_b , λ'_{ab} and λ'_c . In general, μ gets split at two levels. First, among the n join operators (say $\mu_{\bowtie_1}, \mu_{\bowtie_2}, \dots, \mu_{\bowtie_j}, \dots, \mu_{\bowtie_n}$). Then, for each join \bowtie_j , μ_{\bowtie_j} is divided among each of its respective half-way joins μ_{\bowtie_j} and μ_{\bowtie_j} .

Freshness of Multi-Join Results. When the resources become limited, the produced query results may no longer be perfectly *fresh*, as in Query Q1 (Fig. 1). The result freshness is further compromised by throughput optimizing resource allocation to highly productive half-way joins. Consequently, little or no resources are left for the less productive components of the plan. Therefore, under a *throughput optimizing* scheme, insufficient scheduling of certain operators may lead to their *starvation*. Moreover, when a *starved* upstream operator does not produce sufficient intermediate results, the dependent join state in the downstream operator tends to become *stale*. The join results produced using such stale states are also stale, thus further deteriorating the *result freshness*. In Fig. 4b, if $(c \bowtie S_{AB})$ is most productive, the assignment of complete μ to λ'_c would *starve* \bowtie_1 , leading to the *staleness* of the state S_{AB} and eventually also to that of the final query results.

Definition 1. The *freshness* predicate, namely, \mathbb{F}^l for a stream I , requires that joined tuples produced beyond time T must not contain stream I tuples with arrival times older than $|T - \mathbb{F}^l|$.

Under CPU-limited execution, the user can supply a *freshness* predicate \mathbb{F}^l for each stream I (Def. 1). The type of the *freshness* and the window predicate must be the same,

³ A fine-grained *time correlation-awareness* [9] can be used for subset selection along with JDA.

⁴ For simplicity σ_i denotes overall selectivity of \bowtie_i ; in reality each half-way join has an associated selectivity as in Fig. 4a.

i.e., time or count-based. By default the freshness predicate \mathbb{F}^I equals the WINDOW predicate when the users require the results to be 100 % fresh. Query results not fulfilling the *freshness* predicate are considered stale and thus useless. *In this work we focus on achieving maximal throughput while satisfying the user-defined freshness specification.*

3 The JDA Technique

The state-of-the-art JDA technique uses a *half-way join productivity-based* optimization. Here, we define the *half-way join productivity* ρ_h metric (Def. 2) and present the JDA policy.

Definition 2. *The productivity of the half-way join $\bowtie_i \equiv (i \bowtie S_j)$, denoted by $\rho_h(i \bowtie S_j)$, is the throughput contribution (r_{\bowtie_i}) of \bowtie_i per input tuple processed by \bowtie_i .*

$$\rho_h(i \bowtie S_j) = \frac{r_{\bowtie_i}}{\lambda'_i} = \sigma_j \times |S_j| \quad (4)$$

JDA Policy: *Allocate available resources μ starting with the most productive half-way join until μ gets exhausted.*

In a single join operator (Fig. 4a), the μ resources (= 300 tuples/sec) must be divided among the *probe allowances* (here λ'_a and λ'_b), such that the throughput r_{\bowtie} of $A \bowtie B$ is maximized. By Equation 4 the ρ_h of half-way joins are: $\rho_h(a \bowtie S_B) = \sigma_B \times |S_B| = 0.005 \times 500 = 2.5$ and $\rho_h(b \bowtie S_A) = \sigma_A \times |S_A| = 0.001 \times 5000 = 5$ joined tuples/input tuple/sec. For $\mu = 300$ tuples/sec, JDA achieves a throughput of $r_{\bowtie} = 300 \times 5 = 1500$ by assigning all of μ to λ'_b and none to λ'_a .

Applying JDA to a Multi-Join Plan. Now, we derive a variant of the JDA policy, called JDA_M , to make it applicable to the multi-join plans. It is defined as follows:

JDA-MultiJoin Policy: *Allocate μ in two levels, namely,*

1. *Divide μ equally among all n join operators, so for join \bowtie_j ($\forall j = 1, 2, \dots, n$), $\mu_{\bowtie_j} = \frac{\mu}{n}$.*
 2. *Within each operator \bowtie_j , apply BestHJP to assign all μ_{\bowtie_j} towards the most productive half-way join component of \bowtie_j .*
-

In Figure 5, JDA_M first splits the μ equally among the two joins \bowtie_1 and \bowtie_2 , i.e., 150 tuples per second are allocated to each join. In the next step, JDA is applied locally within each operator. In \bowtie_1 , $\rho_h(b \bowtie S_A) > \rho_h(a \bowtie S_b)$, so $(b \bowtie S_A)$ is assigned the 150 tuples per second. Similarly, in \bowtie_2 , $\rho_h(ab \bowtie S_C) > \rho_h(c \bowtie S_{AB})$, thus $(ab \bowtie S_C)$ is assigned the 150 tuples per second. An estimated query throughput of 750 joined tuples per second is achieved (Equation 3). We observe here that while \bowtie_1 produces 750 tuples per second, only 150 out of those can actually be used to probe the partner join state in \bowtie_2 . Hence, there is an **over-utilization** of resources in \bowtie_1 .

The local nature of the JDA technique and its failure in producing optimal throughput in a multi-join plan motivates us to explore operator interdependencies for solving the identified problems as described next in Section 4. In our experiments (Section 5) we compare the JDA_M strategy against our proposed approach.

4 The Proposed JAQPOT Approach

4.1 Optimizing Throughput in Multi-join Queries

Throughput optimization in a *multi-join* plan requires **producer-consumer matches** between every successive join pair where all intermediate tuples produced by a *producer* join must get consumed by the downstream *consumer* join for probing the partner join state. Based on this insight, we propose a synchronized plan level resource allocation strategy.

Input Paths. We introduce the notion of *input paths* in Def. 3.

Definition 3. Given a multi-join plan Q with k input streams⁵ ($I = 1, \dots, k$); each pipeline of half-way joins from the leaf to the root operator forms an **input path** denoted by Path^I . A path having n join operators between input stream I and the output of query is called an **n -hop path**.

In our example plan (Fig. 4b), we identify three input paths, namely, Path^A , Path^B and Path^C . Path^A , a 2-hop path, is composed of two sequential half-way joins, namely, $(a \bowtie S_B)$ followed by $(ab \bowtie S_C)$, also written as $(a \bowtie S_B \bowtie S_C)$. Along an n -hop Path^I , every input tuple joins and propagates through n half-way joins upto the root. Similar to the half-way join productivity (ρ_n), we now define the *cardinality of intermediate joined tuples*, denoted as ϕ_j^I , produced by the j^{th} half-way join along Path^I . As in Eq. 5, ϕ_j^I is computed by multiplying the productivities (ρ_n) along Path^I upto j . Here, superscript p denotes the *partner* join state. ϕ_j^I forms an important component of the core formulae that we define next.

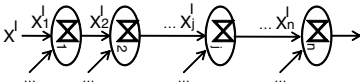


Fig. 6. Division of X^I resources within Path^I

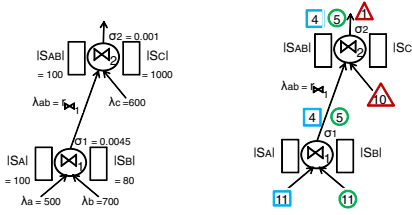
$$\phi_j^I = \prod_{g=1}^j (\sigma_g^p \times |S_g^p|). \quad (5)$$

$$X_j^I = X_1^I \times \phi_{j-1}^I = \left(\frac{X^I \times \phi_{j-1}^I}{1 + \phi_1^I + \dots + \phi_{n-1}^I} \right). \quad (6)$$

Division of Resources within an Input Path. X^I of the total μ resources are allocated to an n -hop path Path^I . Figure 6 depicts the division of X^I among the half-way joins of Path^I as *probe allowances* $X_1^I, X_2^I, \dots, X_j^I, \dots, X_n^I$. For an effective division of X^I **producer-consumer matches** must be established between every pair of successive half-way joins, such that the output of a producer equals the *probe allowance* of the consumer join. Each such *probe allowance* for the j^{th} half-way join, denoted as X_j^I , is expressed in terms of X^I as in Eq. 6. Here, by Eq. 5, ϕ_{j-1}^I denotes the *cardinality of the $(j-1)^{\text{th}}$ intermediate joined tuples*.

Path Productivity. We now establish a *novel* metric that **measures the contribution of an input path to the overall query throughput** (Def. 4).

⁵ If stream I is used multiple times as input to the plan (self-joins), then separate copies of I will be used as separate inputs.



(a) Parameter assignment (b) Path-wise throughput

Fig. 7. Example 2-join plan

Path ^I	Path ^A	Path ^B	Path ^C
Resources used (X ^I)	15	16	10
Output rate r _{⋈^Iroot} ^I	4	5	1
Path Productivity ρ _p (Path ^I)	0.266	0.312	0.1

Fig. 8. Path productivity table

Definition 4. The *path productivity* of Path^I, denoted by ρ_p(Path^I), is its contribution to the query throughput *per tuple t processed*⁶ *within Path^I*.

$$\rho_p(\text{Path}^I) = (\sigma_n^p \times |S_n^p|) \times \left(\frac{\phi_{n-1}^I}{1 + \phi_1^I + \dots + \phi_{n-1}^I} \right) = \left(\frac{\phi_n^I}{1 + \phi_1^I + \dots + \phi_{n-1}^I} \right). \quad (7)$$

If X^I resources are assigned to Path^I, the component X_n^I assigned to the root half-way join may be computed using Eq. 6. The throughput contribution by processing the X^I resources along Path^I, denoted as (r_{⋈^Iroot}^I) = (σ_n^p × |S_n^p|) × X_n^I. Therefore, by Def. 4, the productivity of Path^I can be computed as ρ_p(Path^I) = $\frac{r_{\text{⋈}^I \text{root}}^I}{X^I}$, as given in Eq. 7.

In our example plan (Fig. 4b), X^B resources allocated to Path^B will be divided among λ'_b and λ'_{ab}. By Eq. 6, an effective division of X^B over Path^B will be λ'_b = (X^B/6) and λ'_{ab} = (5 × X^B/6). Using Eq. 7, the total throughput contribution (r_{⋈^Broot}^B) achieved is estimated as (5 × X^B/6) × (σ_c × |S_c|). Thus, if X^B = 600 tuples/sec, λ'_b gets 100, then producing r_{⋈^B1} = 500 tuples/sec as intermediate output. λ'_{ab} gets 500 tuples/sec (= X^B - λ'_b) and a *producer-consumer match* is achieved between ⋈₁ and ⋈₂.

Discussion. The path productivity ρ_p metric defaults to the notion of half-way join productivity ρ_h (Sec. 3) when applied to a single operator. ρ_h is a *local* operator level metric whereas ρ_p metric establishes the contribution of a complete input path to the *query* throughput r_{⋈^Iroot}^I. The former takes only the tuples directly input into the half-way join into consideration, whereas the ρ_p metric instead considers all the tuples processed anywhere along the path, be it at the leaf, the intermediate and the root operators.

4.2 Path Productivity-Based Join Adaptation

Given plan Q with k input paths, namely, Path^A, Path^B, ..., and Path^k, their path productivities can be computed using Formula 7. The 2-join plan in Fig. 7 may be translated into a *path productivity table* (Fig. 8). This translation is based on the *input rates*, the *selectivities* and the *state sizes* along each path within the plan. For each input path Path^I, the *path productivity table* lists (a.) *the resources used* (X^I), (b.) *the query output rate* (r_{⋈^Iroot}^I) achieved using X^I resources, and (c.) *the path productivity* (ρ_p(Path^I)).

In Fig. 7b) for Path^A, the resources X^A = 15 tuples/sec may be divided across the two half-way joins such that λ'_a = 11 and λ'_{ab} = 4. The throughput contribution of Path^A,

⁶ Tuple t refers to either a leaf or an intermediate tuple in Path^I.

denoted as $r_{\text{root}}^A = 4$ as $\sigma_C = 1$. Thus, for every 15 tuples consumed by Path^A in a second, it will produce 4 t^{ABC} joined tuples. The values in Fig. 8 may be *fractional*. Once a multi-join plan has been translated into a productivity table, our join adaptation problem can be formulated as a variant of the knapsack problem [14], as given below.

Problem 1. Join adaptation knapsack problem: Given a plan Q with k paths $\text{Path}^1, \text{Path}^2, \dots, \text{Path}^k$, when Path^I is assigned resources in multiple M^I of X^I , then its throughput $r_{\text{root}}^I = M^I \times \rho_p(\text{Path}^I)$. By defining a^I to be 1 if Path^I is chosen in a solution and 0 otherwise, we can formulate this **JDA-Knapsack** problem as:

$$\text{Maximize } \sum_{I=1}^k a^I \times r_{\text{root}}^I$$

subject to:

$$\sum_{I=1}^k a^I \times M^I \times X^I \leq \mu.$$

JAQPOT Policy: Allocate μ iteratively to the next most productive path until μ gets completely consumed.

Assume $\mu = 30$ tuples/sec. Using the JAQPOT Policy for the productivity table listed in Fig. 8 the most productive path Path^B will be assigned 16 tuples (out of 30). The remaining 14 resources fall short of $X^A=15$, the minimum resources required by the second most productive path Path^A . Thus, Path^C will be chosen and assigned 10 resources, wasting the remaining 4 resources. The total throughput thus achieved is 6 tuples/cycle (assume each cycle runs for 1 second). A more effective assignment would be to instead give the complete 30 ($=15 \times 2$) tuples/cycle to Path^A and achieve 8 ($=4 \times 2$) tuples/cycle as throughput. This illustrates that a **greedy application of the JAQPOT Policy fails to achieve optimal throughput**.

Above, we find ourselves working under rigid constraints. First, each execution cycle runs *independently* of its predecessor and successor execution cycles. Second, we assume a *discrete execution model* where X^I, r_{root}^I and μ must be *whole numbers*. Under this model the *throughput optimization* problem does not exhibit the *greedy choice property* ([14]). Thus, a *dynamic programming* knapsack solver must be employed to achieve an assignment yielding optimal throughput which runs in $O(k \times \mu)$, for k input streams and μ available computing resources. For higher values of μ , this solver would be extremely compute-intensive. Therefore, we now explore alternate greedy strategies for solving this problem.

The Greedy Knapsack Solver. We now relax the above restrictions. First, instead of *independent execution cycles*, each being assigned *distinct* μ resources, we now consider the *coordinated execution across successive cycles*. For example, two successive cycles producing 3 and 2 join tuples respectively will result in the overall path productivity $\rho_p(\text{Path}^I)$ to be 2.5 tuples/cycle. As we will see shortly, this achieves even higher output rates than produced under the *discrete* execution model. Once such a group of successive cycles is identified, we can view their combination as a *mega cycle*. Secondly, X^I, r_{root}^I and μ values can now be *fractional*. Thus, for Path^B (Fig. 7), $X^B = 16$ tuples/sec and $r_{\text{root}}^B = 5$ tuples/sec may be re-phrased as Path^B using $X^B = 8$ tuples/sec to produce $r_{\text{root}}^B = 2.5$ tuples/sec. While *fractional* tuples cannot be consumed (or produced) in a single cycle, over the span of multiple successive cycles a virtual consumption (or production) of *fractional* tuples per cycle may arise.

These relaxations are *mutually complementary* and their benefit is twofold. First, multiple cycles may be scheduled together. Second, as fractional resource assignment is allowed, high productivity paths consuming resources X^I greater than μ , that would otherwise be eliminated in the *discrete model*, may now be assigned resources. Also in a real-world CPU-limited scenario, the resources are more likely to be an estimated μ value available over a duration spanning multiple cycles rather than being a *distinct* μ value available to each cycle. Under this *continuous execution model*, our *JDA-Knapsack Problem* [1] now exhibits both the *greedy choice property* and the *optimal substructure property* [14]. This implies that we can now use a *greedy knapsack solver*, henceforth referred to as **Greedy Path Productivity-based Multi-Join Adaptation (GrePP)**, to tackle our problem.

For our running example in Fig. 7, GrePP selects the most productive path, Path^B (Fig. 8), and allocates all of μ (=30 tuples/sec) such that λ'_b gets 20.62 tuples/sec and λ'_{ab} gets 9.38 tuples/sec. The estimated query throughput $r_{\text{root}}^{\text{GrePP}}$ is 9.38 tuples/sec and is greater than that in the *discrete model*. It is guaranteed to be *optimal* [14]. **GrePP runs in $\mathcal{O}(k \log(k))$ time [14] for a plan joining k streams and thus is independent of μ .**

4.3 Satisfying Freshness in Multi-join Queries

The throughput optimizing allocation by GrePP may still suffer from *result staleness*. We allow users to supply the *freshness* predicates (Def. 1) for each input stream. Now, we extend our **path-productivity based** model to incorporate this notion of *freshness*.

The key idea here is that the **freshness predicates** defined over streams are fulfilled by translating them into **refresh rates** for the join states inside the plan. By Def. 1, tuple t^I from stream I must not be part of the joined results beyond time $(t^I.ts + \mathbb{F}^I)$, where $t^I.ts$ denotes the arrival time of t^I . To enforce this constraint, every tuple t^I from stream I and all its intermediate joined tuples must be purged from the plan by $(t^I.ts + \mathbb{F}^I)$ time (or tuple for count-based freshness). In Fig. 9a, stream C contributes the singleton t^C , the intermediate t^{CD} and t^{CDE} tuples, get stored in *own* states S_C , S_{CD} and S_{CDE} , respectively. State S_C gets refreshed by incoming tuples at λ_c tuples/sec, whereas the intermediate states S_{CD} and S_{CDE} get refreshed with tuples t^{CD} and t^{CDE} at a rate dependent on the portion of μ allocated to λ'_{cd} and λ'_{cde} , respectively. Such intermediate states, such as S_{CD} and S_{CDE} , are called *staleness susceptible states* (highlighted in Fig. 9a). In a steady stream, $\frac{S_C}{\lambda_c}$, $\frac{S_{CD}}{\lambda'_{cd}}$ and $\frac{S_{CDE}}{\lambda'_{cde}}$ denote the time duration for which a singleton t^C and its corresponding t^{CD} and t^{CDE} tuples will remain in their respective *own* states S_C , S_{CD} , S_{CDE} . To satisfy the predicate \mathbb{F}^C for stream C, $\mathbb{F}^C \geq (\frac{S_C}{\lambda_c} + \frac{S_{CD}}{\lambda'_{cd}} + \frac{S_{CDE}}{\lambda'_{cde}})$.

Lemma 1. *To fulfill the freshness predicate \mathbb{F}^I for stream I, $\mathbb{F}^I \geq \sum_{j=1}^n \frac{S_j^o}{\lambda'_j}$, where λ'_j and S_j^o denote the **probe allowance** and the **own join state**, respectively, at j^{th} operator along Path^I , storing intermediate tuples having t^I tuples.*

Using Lemma 1, the *freshness* predicate \mathbb{F}^I on any stream I can only be fulfilled by allocating sufficient resources λ'_j to each operator j along Path^I so that its *own* join states

get refreshed at sufficient rates. Our model of input paths enables us to gain further insights into the *result staleness* problem. We observe that each input path contains one or more of these *staleness susceptible states*. Moreover, each susceptible state, say S_{CD} , may receive input from multiple paths. For example, S_{CD} gets input from $(c \bowtie S_D)$ and $(d \bowtie S_C)$. Also S_{CDE} gets input from $(cd \bowtie S_E)$ and $(e \bowtie S_{CD})$. Thus, *staleness susceptible states* may be refreshed *synchronously* by allocating resources to the paths covering those states. The *freshness* predicates can be satisfied by fulfilling the corresponding *refresh rates* of a half-way join (Def. 5) of each *staleness susceptible state*, i.e, if the input rate λ'_j (λ_j for leaf) at each half-way join exceeds the desired R_{S_j} .

Definition 5. The *refresh rate* R_{S_j} of a state S_j , denotes the minimum number of new tuples required to be inserted per time unit into state S_j to prevent it from becoming stale. A *staleness susceptible state* S_j is said to be covered if its refresh rate R_{S_j} is fulfilled.

Given the foundation, in Problem 2 we translate the *freshness* satisfiability problem into a **weighted multiple set cover problem (WMSCP)** [18] over the *staleness susceptible states*. Given the set of all *staleness susceptible states* and the input paths that include those states, the goal is to identify the set of paths, called the *minimal coverage paths*, that cover all the *staleness susceptible states* utilizing the *minimum* computing resources $\Delta\mu$ out of the total μ resources. The remaining $(\mu - \Delta\mu)$ resources are used by GrePP for throughput optimization. In Fig. 9a, Path^A and Path^C are such *minimal coverage paths* covering all staleness susceptible states in \bowtie_3 and \bowtie_4 .

Problem 2. Coverage of staleness susceptible states as a weighted multiple set cover problem (WMSCP): The Universe U consists of m staleness susceptible states $= S_1, S_2, \dots, S_m$ with required refresh rates $= R_{S_1}, R_{S_2}, \dots, R_{S_m}$, respectively. The k input paths $P = Path^1, Path^2, \dots, Path^k$ cover all the staleness susceptible states where $\cup_{l=1}^k Path^l = U$ such that each path $Path^l$ has a positive real cost (resources used) X^l . If a n -hop $Path^l$ contains state S_j , then the resources used for S_j in $Path^l$ are denoted as X_j^l , such that for the n states of $Path^l$ $\sum_{j=1}^n X_j^l = X^l$.

A k -tuple $M = M_1, M_2, \dots, M_k$ constitutes a multiple cover for U in which the number of times state S_j is covered is defined to be the sum of M_l 's for those $Path^l$'s which contain S_j . Total weight of the **multiple cover** is defined as $\sum_{l=1}^k X^l \times M_l$. WMSCP seeks the minimum weight multiple cover for U such that every state S_j is covered for at least its refresh rate R_{S_j} . By defining b_j^l to be 1 if $S_j \in Path^l$ and 0 otherwise, we can now write our WMSCP problem as:

$$\Delta\mu = \text{Minimize } \sum_{l=1}^k X^l \times M_l$$

subject to:

$$\sum_{l=1}^k b_j^l \times X_j^l \times M_l \geq R_{S_j}, \forall j = 1, 2, \dots, m.$$

Complexity and Optimality Analysis. WMSCP is strongly NP-Hard and the cost of an optimal solution may be too high for our dynamically scenario. Thus, we use a greedy algorithm called *GH-WMSCP* [18]. We use GH-WMSCP to satisfy the refresh rates and in turn to fulfil *freshness* predicates. The time complexity TC(GH-WMSCP)

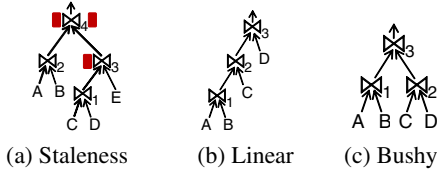


Fig. 9. Multi-join plans

Input: Path productivity table $\tau_p[][1..k]$ for plan Q , refresh rates $R_{S_{1..j}} \forall j$ states, available resources μ

Output: Assignment of μ to plan Q $PathAssign[][]$

```

1:  $PathAssign[ ][ ] \leftarrow GH\text{-}WMSCP(\tau_p[ ][1..k], R_{S_{1..j}})$ 
2:  $\Delta\mu \leftarrow \sum_{i \rightarrow 1}^k PathAssign[i][ ]$ 
3:  $PathAssign[ ][ ] \leftarrow GrePP(\mu - \Delta\mu, \tau_p[ ][1..k])$ 
4: return  $PathAssign[ ][ ]$ 

```

Fig. 10. The JAQPOT Algorithm

$= \mathcal{O}(m \times k + m^2)$ for m staleness susceptible states and k input paths. The cover found by GH-WMSCP will at most differ from the optimal cover for WMSCP, denoted as $OPT(WMSCP)$, by a factor of $\ln(m)$ [18].

Lemma 2. For k input streams in a join query Q , there are exactly $(k-2)$ staleness susceptible states irrespective of the query shape, be it linear, semi-bushy or bushy.

Lemma 2 relates the counts of the input paths (k) and the susceptible states (m) in a query. For example, in both the plans, namely, *linear* (Fig. 9b) and *bushy* (Fig. 9c), $k=4$ and $m=2$. Therefore, substituting m with $(k-2)$ in the expression for the time complexity of GH-WMSCP, $TC(GH\text{-}WMSCP) = \mathcal{O}(k^2)$.

4.4 The Integrated JAQPOT Algorithm

We now present our algorithm called *Join Adaptation at Query plan-level using Path-productivity for Optimizing Throughput*, in short, JAQPOT (Fig. 10). JAQPOT first assigns a fraction $\Delta\mu$ out of μ available resources towards fulfilling the freshness requirements using the GH-WMSCP. Further, the greedy knapsack solver GrePP achieves an optimal query throughput using the remaining resources $(\mu - \Delta\mu)$. JAQPOT returns the join adaptation assignment in $PathAssign[][]$, where $PathAssign[I][j]$ denotes the resources assigned to the j^{th} half-way join of $Path^I$. The overall time complexity of our solution $TC(JAQPOT) = TC(GH\text{-}WMSCP) + TC(GrePP) = \mathcal{O}(k^2 + k \times \log(k)) \simeq \mathcal{O}(k^2)$. Thus, **JAQPOT runs in quadratic time of k** irrespective of the plan shape.

Run-time Query Adaptation. An initially optimal resource allocation by JAQPOT may become sub-optimal due to the dynamic nature of the streams. Thus, we adopt a simple yet effective strategy for runtime adaptation (details in [13]). We measured the runtime overheads of JAQPOT and found that GH-WMSCP incurs the highest cost. Performance of GH-WMSCP for different parameters, such as sizes of input and sets, has been thoroughly studied in [18]. Thus, we instead focus our experimental study on throughput and freshness.

⁷ We chose to satisfy the freshness predicates while optimizing throughput as this adaptation is sufficient for real world applications. We found in our experimental study (Sec. 5) that in practice realistic freshness predicates are indeed fulfillable using only a small share of the resources.

5 Experimental Evaluation

We now examine the effectiveness of JAQPOT compared against the state-of-the-art JDA technique (described in Section 3). We implemented JAQPOT and JDA_M within the CAPE stream engine [15].

Parameter	Value
Arrival rates (λ_i)	300 ~ 1200 tuples/sec
Window sizes of states ($ S_I $)	200 ~ 5000 tuples
Join selectivities (σ_i)	0.01 ~ 0.1
Available Resources (μ)	0 ~ 100 % of saturation
Freshness predicates (F^I)	$1.5\times \sim 5\times S_I $

Fig. 11. Experimental parameters with values

Objectives. The goal of our experimental study is to substantiate the observations of our analytical study (Sec. 4) that JAQPOT is capable of producing *near optimal throughput* together with maintaining *result freshness*. We evaluate JAQPOT and its competitor JDA_M policy by measuring their performance in (a) *producing query throughput*, and (b) *fulfilling the freshness predicates*. We examine the following critical questions with focus on the two performance measures:

- *What is the impact of stream and query parameters, such as λ_i , σ_i , and S_I as well as the query shape on the throughput produced by the JDA techniques?*
- *How does the throughput produced by JAQPOT and JDA_M techniques compare with the saturation throughput⁸ with change in μ ?*
- *In the absence of the set-coverage solver (GH-WMSCP), how badly do the throughput optimization techniques perform in terms of the result freshness?*
- *In JAQPOT, what fraction of resources get assigned for fulfilling freshness as opposed to achieving high throughput?*

Experimental Setup. All experiments are run on a machine with Java 1.6 runtime, Windows 7 with Intel(R) Core(TM)2 Duo CPU@2.13 GHz processor and 4 GB RAM. All techniques are tested rigorously using synthetic streams and distinct query shapes with arbitrary parameter settings (Table 11). Further, the applicability to a real-world application is also verified using the Weatherboards data set [1]. The results for the experiments with the real data set are not included in this paper. Please refer to technical report [12] for details.

5.1 Throughput Production in Synthetic Data

The goal of our experiments is to compare the throughput produced by both the JDA techniques under (a) fluctuating streams, and (b) changing resource availabilities. We

⁸ The minimum total resources required to process the full query workload with no CPU limitation are called the *saturation resources*. The corresponding throughput produced is called the *saturation throughput*.

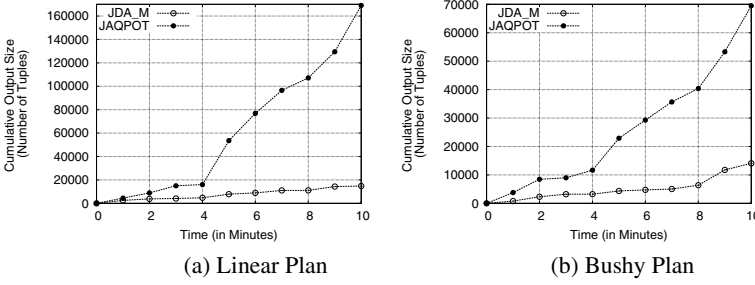


Fig. 12. Impact of fluctuations in streams

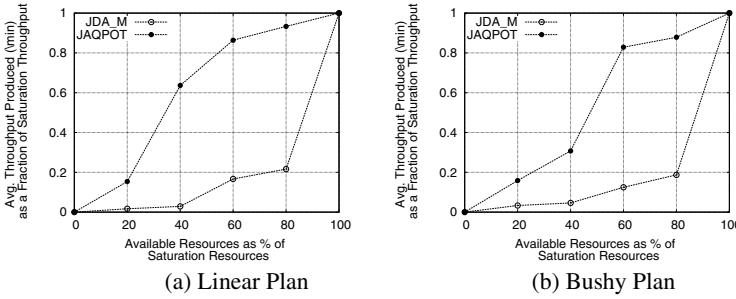


Fig. 13. Impact of resource availability

measure throughput as *the cumulative join output tuples produced over time*. We use an equi-join of 4 streams, namely, A, B, C and D with two different query shapes, namely, *linear* (Fig. 9.b) and *bushy* (Fig. 9.c). While the join order of the *linear* plan is $((A \bowtie B) \bowtie C) \bowtie D$, that of the *bushy* plan is $((A \bowtie B) \bowtie (C \bowtie D))$. The data streams are generated according to the *Poisson* distribution that models the arrival pattern of several real-world stream applications. Overall, a variety of scenarios are evaluated by changing the λ_i , σ_i and S_I parameters for each query shape (Fig. 11).

Impact of Fluctuating Stream Parameters. The fluctuating input streams are simulated by changing *operator selectivities*. The *window sizes* and *arrival rates* were observed to have *similar* effects on the workload as that of the selectivities, thus we omit them here. Query workloads can be adjusted by generating streams such that the join selectivities become high (or low) as desired. Here, we fixed the μ to 30% of saturation whereas F^I is set to $1.5 \times \text{WINDOW}$ predicates on each stream I.

In Fig. 12, we measure the cumulative throughput (y-axis) as time progresses (x-axis) for a total of 10 mins of steady state query execution. In the *linear* plan (Fig. 12.a), the selectivities first change at 3 mins. from SEL1 ($\bowtie_1 = 0.01 \mid \bowtie_2 = 0.01 \mid \bowtie_3 = 0.05$) to SEL2 ($\bowtie_1 = 0.03 \mid \bowtie_2 = 0.03 \mid \bowtie_3 = 0.05$) and further at 7 minutes from SEL2 to SEL3 ($\bowtie_1 = 0.03 \mid \bowtie_2 = 0.03 \mid \bowtie_3 = 0.1$). From SEL1 to SEL2, the selectivities of \bowtie_1 and \bowtie_2 *triple* while keeping \bowtie_3 constant. From SEL2 to SEL3, the selectivity of the root \bowtie_3 *doubles* while the selectivities of \bowtie_1 and \bowtie_2 remain unchanged. This change in the root operator \bowtie_3 improves the throughput production as by JAQPOT even

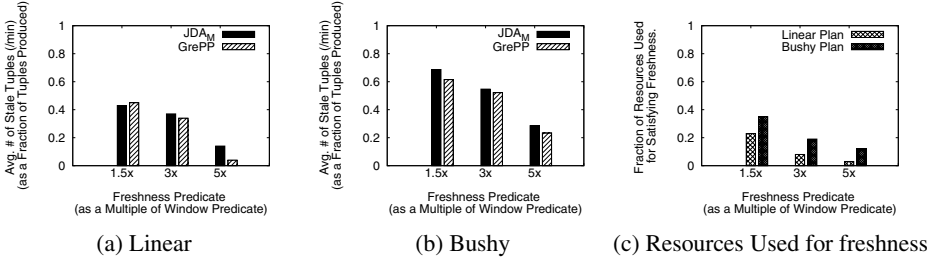


Fig. 14. Evaluation of freshness predicates

more significantly than the change in non-root operators. The performance of the JDA_M strategy is significantly lower than JAQPOT. Figure 12b illustrates the results for the *bushy* plan with changes in the selectivities at 3 and 7 mins, just like the linear plan. Here JAQPOT again produces high throughput while JDA_M continues to produce output at a very low rate.

Impact of Changing Available Resources. These charts (Fig. 13) summarize the performance of the techniques over the complete range of μ values from 0% to 100% of saturation. A variety of parameter settings are used, as in Table 11. The charts depict μ as a percentage of the *saturation resources*. On the y-axis, the throughput produced by JAQPOT and JDA_M strategies, averaged over several runs, is shown as a percentage of the saturation throughput.

For the linear plan (Fig. 13a) JAQPOT outperforms the JDA_M strategy producing more than 80% of the saturation throughput while using only 60% of the resources. Averaged over the different μ values, JAQPOT consistently beats the JDA_M strategy by producing 3 \times as many tuples/min on average, with a maximum of 6.5 \times at 40% of saturation resources.

For the bushy plan (Fig. 13b), the trends are similar. Overall, JAQPOT performs much better for the linear plan than the bushy plan. In the linear plan, all the *staleness susceptible states* can be synchronously refreshed as they are covered by fewer (possibly single) path, whereas, for the bushy plan, atleast two paths require resources for *freshness* fulfillment, thus leaving fewer resources for throughput optimization.

5.2 Evaluating Result Freshness

The purpose of these experiments is twofold, namely, (a) to establish that result staleness is aggravated by the JDA approaches, and (b) to analyse how much of the resources are spent in satisfying *freshness*. The staleness of results is measured by counting the number of tuples produced that violate a given *freshness* predicate \mathbb{F}^f as per Def. 11.

Result Staleness in Join Adaptation. Next, we substantiate our hypothesis that the *throughput optimizing schemes aggravate the result staleness problem*. We compare the JDA_M and the GrePP knapsack solver omitting the GH-WMSCP component such that GrePP produces stale tuples in the absence of GH-WMSCP. We perform these experiments on the linear and bushy plans (Figs. 9b and c). For each plan, we create many scenarios by varying the parameter settings (Table 11). Here, μ is fixed at 300.

We evaluate three distinct settings of the *freshness* predicate, namely, $1.5\times$, $3\times$, and $5\times$ of the window size. The higher the freshness predicate, the more tolerant the user query is to staleness. For each freshness value, we count the number of stale tuples produced by each technique. The three *freshness* predicate values (x-axis) are plotted against the average fraction of stale tuples/min (y-axis).

For the linear plan (Fig. 14a), as the freshness predicate is relaxed from $1.5\times$ to $5\times$, there is a marked drop in number of stale tuples. The JDA_M strategy produces high amounts of stale tuples. In absence of GH-WMSCP, even GrePP produces a substantial amount of stale tuples. The trend is similar in the bushy plan (Fig. 14b). However, an even larger number of stale results are produced in the bushy plan as compared to the linear plan.

Resource Utilization for Satisfying Freshness. We aim to evaluate the fraction of the available resources allocated by JAQPOT towards freshness fulfillment. For these experiments, we run JAQPOT (GH-WMSCP + GrePP) for several settings of the linear and the bushy plans by changing the query parameters, including different μ values. We again evaluate three distinct *freshness* settings, namely, $1.5\times$, $3\times$, and $5\times$ of the window predicates.

In Fig. 14c, the freshness predicate (x-axis) is plotted against the fraction of resources used for satisfying freshness. We find that as the freshness predicate is relaxed, the demand for resources requirement also reduces drastically. For freshness tolerance of $5\times$, the linear plan utilizes only 3% and 5% resources for freshness, respectively. Further, as the bushy plan faces higher risk for staleness, the bushy plan uses significantly larger portions of resources for freshness (35% for $1.5\times$). Thus, most reasonable freshness predicates may be fulfilled by allocating less than 10% of μ on average. Moreover, our proposed solution is guaranteed to find the best solution, if one exists. The proof can be easily implied from optimality of the set-cover [18] and the knapsack [14] solvers.

Infeasible plans. Among the plans we evaluated, we periodically found some *infeasible* plans, i.e., whose freshness predicates were not achievable under existing conditions. In particular, about 8% of the evaluated plans were infeasible, 85% of which were for the rigid freshness predicate $1.5\times$ and 65% were for the bushy plans.

Experimental Conclusions. The findings of our experimental study are:

1. JAQPOT continuously produces near-optimal throughput even in bursty streams.
2. JAQPOT consistently outperforms the state-of-the-art ρ_h -based JDA_M policy by producing 2~6 times higher throughput for all tested cases.
3. JAQPOT performs better in *linear* plans compared to *bushy* plans, as bushy plans utilize more resources for *freshness* fulfillment.
4. In CPU-limited processing, *result staleness* problem is further aggravated if throughput optimizing techniques are employed.
5. For all satisfiable cases, JAQPOT guarantees a throughput optimizing allocation.

6 Related Work

Load shedding [4, 16] is popular in CPU-limited scenarios. Shedding directly drops the tuples from the streams and the data is permanently lost. Shedding solutions, with

an exception of [4, 16] focus on optimizing a *single* join operator or a single MJoin operator [9]. Tatbul et al. [16] first applied load shedding to streaming databases. As indicated in [16], they *do not* address the additional issues related to processing windowed joins over streams. Ayad et al. [4] propose static optimization and in the absence of a feasible plan they pick a plan augmented with shedding operators placed on the input streams to make it feasible. GrubJoin [9] targets the MJoin operator by leveraging *time correlation-awareness*. It focuses on a *single* MJoin operator, whereas our work tackles an orthogonal problem of operator interdependencies within a plan. Although MJoins utilize less memory, they are typically computationally expensive [17] and are less likely to be selected by the query optimizer in a CPU-limited scenario.

Closest to our work, *join direction adaptation* (JDA) [8, 10] explores the *half-way join productivity* to *selectively allocate* computing resources to maximize the output rate. They focus on a *single* join operator only. In this work, we establish that such traditional JDA technique becomes ineffective for multi-join queries. Further, all these approaches typically address a single optimizing function. None of these approaches focus on leveraging the inter-operator dependency to adapt to run-time fluctuations nor do they consider *result staleness*. Whenever a query with interconnected join operators is used, our solution leveraging operator *interdependency* can be applied in conjunction with the existing approaches [9, 17].

While **operator scheduling** [6, 7] tends to allocate resources at the **coarse** granularity of a query operators, we focus on adaptation at a **finer** granularity of half-way joins within an operator for optimizing throughput. Our work utilizes an adaptive query processing [13] framework for adjusting the join direction of the query plan at run-time.

7 Conclusion

This paper addresses the CPU-limited execution of multi-join queries using join direction adaptation. We propose the *path productivity* metric that leverages the *operator interdependencies* instead of localized operator-centric optimization. We identify *result staleness* as a pressing issue under CPU limitations, and throughput optimizing techniques further aggravate it. Our key contribution is the integrated JAQPOT algorithm that tackles the *result staleness* problem while producing *optimal* query throughput. We validate our analytical findings using experimental studies with both synthetic and real data.

Acknowledgements. We are grateful to Song Wang, Luping Ding and other DSRG members for their efforts in building the CAPE system. We thank Prof. Murali Mani and the anonymous reviewers for their insightful comments.

References

1. Weatherboards dataset from intel berkeley research lab, <http://db.csail.mit.edu/labdata/labdata.html>
2. Abadi, D.J., Carney, D., et al.: Aurora: a new model and architecture for data stream management. VLDB 12(2) (2003)

3. Arasu, A., et al.: The cql continuous query language: semantic foundations and query execution. *VLDB* 15(2) (2006)
4. Ayad, A., Naughton, J.F.: Static optimization of conjunctive queries with sliding windows over infinite streams. In: *SIGMOD* (2004)
5. Babcock, B., Babu, S., et al.: Models and issues in data stream systems. In: *PODS* (2002)
6. Babcock, B., Babu, S., et al.: Chain: operator scheduling for memory minimization in data stream systems. In: *SIGMOD* (2003)
7. Carney, D., et al.: Operator scheduling in a data stream manager. In: *VLDB* (2003)
8. Gedik, B., et al.: Adaptive load shedding for windowed stream joins. In: *CIKM* (2005)
9. Gedik, B., Wu, K.-L., et al.: Grubjoin: An adaptive, multi-way, windowed stream join with time correlation-aware cpu load shedding. *IEEE TKDE* 19 (2007)
10. Kang, J., Naughton, J., et al.: Evaluating window joins over unbounded streams. In: *ICDE* (2003)
11. Liu, B., Zhu, Y., Rundensteiner, E.: Run-time operator state spilling for memory intensive long-running queries. In: *SIGMOD* (2006)
12. Mukherji, A., Rundensteiner, E.A.: Tr-wpi-cs-11-01: Achieving high freshness and optimal throughput in resource-limited execution of multi-join continuous queries: The jaqpot approach. Technical report, WPI (2011)
13. Nehme, R.V., Rundensteiner, E.A., Bertino, E.: Self-tuning query mesh for adaptive multi-route query processing. In: *EDBT* (2009)
14. Pisinger, D.: Where are the hard knapsack problems? *Comput. Oper. Res.* 32(9) (2005)
15. Rundensteiner, E.A., et al.: Cape: Continuous query engine with heterogeneous-grained adaptivity. In: *VLDB* (2004)
16. Tatbul, N., et al.: Load shedding in a data stream manager. In: *VLDB* (2003)
17. Viglas, S., Naughton, J., et al.: Maximizing the output rate of multi-way join queries over streaming information sources. In: *VLDB* (2003)
18. Yang, J., Leung, J.Y.-T.: A generalization of the weighted set covering problem. *Naval Research Logistics* (2005)

Mining Image Databases by Content

Gerald Schaefer

Department of Computer Science, Loughborough University, Loughborough, U.K.

Abstract. Visual information is becoming more important and at a rapid rate. However, creators and users are reluctant to annotate visual content making it difficult to search these collections. Content-based image retrieval (CBIR) techniques extract visual descriptors directly from image data and can hence be used in situations where textual information is not available. In this paper, we give a brief introduction on some of the basic colour descriptors that are employed in CBIR.

1 Introduction

While personal image collections may contain thousands of images, commercial image repositories can comprise several million images [1]. Effective and efficient methods for querying these collections are highly sought after. While images are rarely annotated [2], content-based image retrieval (CBIR) techniques [3], which extract image features describing colour, texture, shape etc. attributes to formulate a query, can be successfully employed. In this paper, we give a brief introduction on some of the basic colour descriptors that are used for CBIR.

2 Content-Based Image Retrieval by Colour

Colour has been shown to be one of the most effective feature types for CBIR. The simplest colour descriptor for CBIR are colour moments [4] which are defined by central normalised moments of the colour distribution of an image (usually mean, standard deviation and kurtosis in each colour channel). Visual (dis)similarity between two images can be described by the L_1 norm between their moment vectors.

Swain and Ballard [5] introduced the use of colour histograms, which record the frequencies of colours in the image, to describe images in order to perform object recognition and image retrieval. As similarity measure they introduced histogram intersection which quantifies the overlap between two histograms and can be shown to be equivalent to an L_1 norm.

Rather than using colour histograms, a more compact descriptor for encoding the colour distribution of images is a colour signature. Colour signatures are a set $\{(c_1, \omega_1), (c_2, \omega_2), \dots, (c_m, \omega_m)\}$ where c_i define colour co-ordinates and ω_i their associated weights (i.e., their frequencies in the image). A common way of deriving colour signatures for images is through a clustering process. Once colour signatures for images are determined, these signatures can be compared by a

metric known as the earth mover's distance [6] which is a flow-based measure and effectively describes the work that is required to transform the colour signature of one image into that of another.

Simple colour features such as colour histograms are fast to compute, and are invariant to rotation and translation as well as robust to scaling and occlusions. On the other hand, they do not carry any information about the spatial distribution of the colours. Colour coherence vectors [7] were introduced as a method of introducing spatial information into the retrieval process. Colour coherence vectors consist of two histograms: one histogram of coherent and one of non-coherent pixels. The L_1 norm is used as the distance metric between two colour coherence vectors.

3 Conclusions

Content-based image retrieval has been a very active research area for the past two decades, and colour features have been shown to be very useful in this context. In this paper, we have given a brief summary of some colour features that are commonly employed for CBIR. While space limitations don't allow us to go into detail here, it should be noted that for efficiency many descriptors (or near equivalents) can also be derived in the compressed domain [8], while colour features are also very useful for image browsing systems which provide an interactive alternative to retrieval-based approaches [9].

References

1. Osman, T., Thakker, D., Schaefer, G., Lakin, P.: An integrative semantic framework for image annotation and retrieval. In: IEEE/WIC/ACM International Conference on Web Intelligence, pp. 366–373 (2007)
2. Rodden, K.: Evaluating Similarity-Based Visualisations as Interfaces for Image Browsing. PhD thesis, University of Cambridge Computer Laboratory (2001)
3. Smeulders, A., Worring, M., Santini, S., Gupta, A., Jain, R.: Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Analysis and Machine Intelligence* 22, 1249–1380 (2000)
4. Stricker, M., Orengo, M.: Similarity of color images. In: Conf. on Storage and Retrieval for Image and Video Databases III. Proceedings of SPIE, vol. 2420, pp. 381–392 (1995)
5. Swain, M., Ballard, D.: Color indexing. *Int. Journal of Computer Vision* 7, 11–32 (1991)
6. Rubner, Y., Tomasi, C., Guibas, L.: The earth mover's distance as a metric for image retrieval. *Int. Journal of Computer Vision* 40, 99–121 (2000)
7. Pass, G., Zabih, R.: Histogram refinement for content-based image retrieval. In: 3rd IEEE Workshop on Applications of Computer Vision, pp. 96–102 (1996)
8. Schaefer, G.: Content-based retrieval of compressed images. In: International Workshop on Databases, Texts, Specifications and Objects, pp. 175–185 (2010)
9. Plant, W., Schaefer, G.: Visualisation and browsing of image databases. In: Lin, W., Tao, D., Kacprzyk, J., Li, Z., Izquierdo, E., Wang, H. (eds.) *Multimedia Analysis, Processing and Communications*. SCI, vol. 346, pp. 3–57. Springer, Heidelberg (2011)

Searching for Complex Patterns over Large Stored Information Repositories*

Nikhil Deshpande¹, Sharma Chakravarthy¹, and Raman Adaikkalavan²

¹ CSE Department, The University of Texas at Arlington

² CIS Department, Indiana University South Bend
{sharma, raman}@cs.iusb.edu

Abstract. Although Information Retrieval (IR) systems, including search engines, have been effective in locating documents that contain specified patterns from large repositories, they support only keyword searches and queries/patterns that use Boolean operators. Expressive search for complex text patterns is important in many domains such as patent search, search on incoming news, and web repositories. In this paper, we first present the operators and their semantics for specifying an expressive search. We then investigate the detection of complex patterns – currently not supported by search engines – using a pre-computed index, and the type of information needed as part of the index to efficiently detect such complex patterns. We use an expressive pattern specification language and a pattern detection graph mechanism that allows sharing of common sub-patterns. Algorithms have been developed for all the pattern operators using the index to detect complex patterns efficiently. Experiments have been performed to illustrate the scalability of the proposed approach, and its efficiency as compared to a streaming approach.

Keywords: Information retrieval, Complex patterns, Document search.

1 Introduction

Although current IR systems [1,2,3,4] are convenient for doing keyword searches, in domains such as federal intelligence, fugitive tracking and searching full-text patent information, there is a need to detect (or search¹) more complex patterns in data sources. Users in these domains may have more precise requirements in terms of what they are searching for. They may be searching for patterns that involve term frequency (e.g., at least 5 occurrences of the phrase “protein clustering”), proximity with sub-patterns (e.g., “peptide” near “saccharide”, in any order, within 5 words of each other), sequence of sub-patterns (e.g., “DNA” followed by “modification”) and so on. Further, the patterns that need to be detected may be arbitrarily complex; that is, they may need to be specified in terms of other patterns (e.g., (“militant” followed by “bomb”) near “Iraq”, separated by 5 positions or less). The expressiveness of search/query specification provided by current IR systems, although satisfactory for general searches, is not adequate for the above application domains.

* This work was supported, in part, by the following NSF grants: IIS-0326505, EIA 0216500, and IIS 0534611.

¹ We use the terms “search” and “detect” interchangeably in this paper.

Detecting complex patterns over text streams has been studied and shown to be possible in [5], in which a suite of complex operators and their algorithms were developed that detect complex patterns over stream data. Detecting patterns over a dynamic text source (e.g., news feeds, IP packets) essentially entails streaming the data to detect the required patterns. In other words, to detect a pattern, the entire data source must be read (or parsed) every time. This is inefficient, but unavoidable, because of the fast-changing nature of the data source. Also, if freshness of the search results are important to the user, it becomes necessary to read the data source every time while processing a query. However, if the data source is relatively static (e.g., Web repositories), it is unnecessary to read the entire source each time a pattern is to be detected. The inefficiency will be exacerbated as the data source grows larger. A better approach would be to build and leverage an index on the data, as is done by search engines, and the information in the index could be used for answering queries. Since the index would be computed off-line, this approach may result in an occasional out-of-date search result. However, considering that the data source is not frequently updated, we can assume this is acceptable to the user. For such relatively static data sources, the gains in terms of efficiency of retrieval that leveraging an index will bring outweigh the slight disadvantage of an occasional out-of-date result.

The techniques developed for searching complex patterns over streams (as in XML streams, news feed, stock prices) in [5] makes use of the sequential inflow of patterns by reading the entire data source to detect a pattern. However, if the same patterns need to be detected in stored data (as in web repositories) then streaming is very inefficient. It is more efficient to index the repository (or use an already existing index) to detect the patterns. Indexing will lose the sequence of occurrence of patterns in the data. This order of occurrence of patterns is the key to detecting patterns based on proximity, containment, sequence, etc. The main contributions of this paper are to: (i) Identify information that is needed as part of the index to correctly and efficiently detect complex patterns as compared to the streaming approach, (ii) Explore the extent of the complexity of the patterns that can be detected using indexed information, and (iii) Develop efficient algorithms for index-based pattern detection.

The rest of this paper is organized as follows: Section 2 discusses the semantics of the InfoSearch operators and Section 3 explains the algorithms used by the operators. Section 4 explains the design of the InfoSearch System. Section 4.1 explains the implementation aspects of the system. Section 5 shows detailed experimental results. Section 6 reviews the related work, and Section 7 concludes the paper.

2 Pattern Specification and Detection

The InfoSearch framework discussed in this paper consists of an expressive query language (introduced in [5]) through which the user can specify patterns and a pattern detection engine capable of using the index to retrieve documents. InfoSearch detailed in this paper has been briefly summarized in [6]. InfoSearch adopts the Pattern Specification Language (PSL) and its associated parser and pattern validator used in InfoFilter [5]. The focus of this paper is on the detection of complex patterns over large document repositories.

2.1 Pattern Specification

An occurrence of a pattern P is the presence of the pattern P in a given document. There is an offset (multiple ones if the pattern occurs multiple times in the document) at which the pattern occurs in the document. O_s is the start offset, and O_e is the end offset of the pattern, where offset is the position of *words* relative to the beginning of the document.

Simple patterns are the basic building blocks and can be either *System-defined* (i.e., pre-defined in the system), or *User-defined*. *Begin_para*, *Begin_document* are examples of system-defined patterns. Examples of simple user-defined patterns are: keywords or phrases.

Complex patterns are composed of simple patterns, complex patterns, and pattern operators (listed below). Any arbitrary complex pattern can be composed using the pattern operators. Current operators supported are summarized below:

OR: Disjunction of two simple or complex patterns P_1 and P_2 , denoted by (P_1 OR P_2), occurs when either P_1 or P_2 occurs. For example, “*information*” OR “*filtering*” will be detected when either one of the keywords occurs.

NEAR: Proximity of two simple or complex patterns P_1 and P_2 , denoted by (P_1 NEAR [D] P_2), occurs when both P_1 and P_2 occur, irrespective of their order of occurrence. “ D ” is the maximum distance allowed between the patterns P_1 and P_2 . Default value of “ D ” is the scope of the operator (which can be the entire document).

FOLLOWED BY: Sequence of two simple or complex patterns P_1 and P_2 , denoted by (P_1 FOLLOWED BY [D] P_2), occurs when the occurrence of P_1 is followed by the occurrence of P_2 in a non-overlapping manner. The end offset of P_1 is less than the start offset of P_2 ; “ D ” is the maximum distance allowed between the two patterns P_1 and P_2 . If the value of “ D ” is 1 (minimum value), this indicates that the patterns P_1 and P_2 form a phrase.

WITHIN: Occurrence of a simple or complex pattern P in the range formed by the start offset of the pattern P_S and the end offset of P_E , denoted by (P WITHIN (P_S , P_E)). The pattern is detected each time pattern P occurs in the range defined by patterns P_S and P_E . For example, “*information filtering*” WITHIN (*BeginPara*, *EndPara*) will be detected whenever the phrase “*information filtering*” occurs within a paragraph. When an expression is specified without a system-defined pattern, the default structure (e.g., a document) is used as the default. User defined P_S and P_E can be used.

NOT: Non-occurrence of a simple or complex pattern P in the range formed by the start offset of P_S and the end offset of P_E . The general specification is (*NOT* [F](P)(P_S , P_E)), where P , P_S , and P_E can be arbitrary patterns. “ F ” indicates the minimum number of occurrences and its default value is 1. For example, *NOT* (“*filtering*”)(“*information*”, “*retrieval*”) will be detected whenever “*information*” is followed by “*retrieval*” without the word “*filtering*” occurring at least once in between them.

FREQUENCY: Multiple occurrences of a simple or complex pattern that exceed or equal to F , denoted by (*FREQUENCY* [F] (P)). A pattern P is detected each time P occurs at least F times, where “ F ” is the minimum number of occurrences specified by the user. The default value of F is 1. All the occurrences that are used for detection should be disjoint (i.e., the end offset of each pattern occurrence should precede the

start offset of the subsequent pattern occurrence). The same set of occurrences will not be used for detecting multiple instances of the same pattern.

SYN: This is an option and is specified along with a single-word pattern (currently), denoted by (P [SYN]), to indicate multiple single-word patterns that have the same meaning, in a succinct manner. Specifying a single-word pattern with SYN option is equivalent to specifying N simple patterns that carry the same meaning (synonyms) as the original pattern. For example, if you specify the word “*bomb*”[SYN] is equivalent to specifying “*bomb*” OR “*explosive device*” OR “*weaponry*” OR “*arms*” OR “*implements of war*” OR “*weapons system*” OR “*munition*” . If any of these words or phrases appears in the text, the pattern “*bomb*”[SYN] is detected. This option adds simplicity and flexibility to the specification of single-word patterns. The same is true for complex patterns with embedded synonym specification, e.g. “*Bomb*”[SYN] NEAR “*Ground Zero*”.

Sample Query: Using the above operators, users can specify complex and meaningful patterns. A complex pattern (“*bomb*” occurring prior to “*ground zero*” occurring twice, with a single occurrence of “*automotive*” or its synonyms), can be specified as:

Pattern P_1 = “*bomb*” FOLLOWED BY “*groundzero*”
 Pattern P_2 = FREQUENCY/2 (P_1)
 Pattern P_3 = P_2 NEAR “*automotive*” [SYN]

2.2 Pattern Detection

Pattern detection semantics are needed for detecting meaningful patterns, since in an unrestricted semantics (where none of the pattern occurrences are discarded after participating in pattern detection) not all the detected patterns are meaningful for an application. Detection semantics essentially delimit the patterns detected and accommodate a wide range of application requirements.

We want to emphasize that we have chosen to define *proximal-unique* semantics in this paper based on the intuition of proximity and disjoint pattern detection. It is certainly possible to define other meaningful constraints leading to other useful semantics. However, the framework remains the same and the algorithms change depending upon the semantics used. It is indeed possible to include semantics of detection as an additional parameter when several of them are defined and supported.

Consider a document containing occurrences of words as shown in Figure 1. Suppose we want to find occurrences of “*cell*” FOLLOWED BY “*nucleic*” within this document. As shown in the figure, there are two occurrences of “*cell*”, one occurring at position 10, say $cell^1$ and the other at position 15, say $cell^2$. The occurrences of nucleic are at position 28 and 41, say $nucleic^1$ and $nucleic^2$ respectively. We could combine either

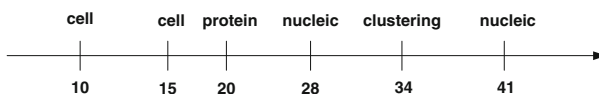


Fig. 1. Pattern Occurrences (Example)

$cell^1$ with $nucleic^1$, or $cell^2$ with $nucleic^1$, or $cell^1$ and $cell^2$ both with $nucleic^1$ as occurrences of the combined pattern “ $cell$ ” FOLLOWED BY “ $nucleic$ ”.

However, it makes more intuitive sense to combine only the closest occurrences, because closely occurring patterns are more likely to be of interest for a search as the correlation here is measured in terms of *proximity*. Hence, we discard the occurrence of $cell^1$ and combine $cell^2$ with $nucleic^1$. In the above example, $cell^2$ is called the *initiator* because it initiates the pattern detection, and $nucleic^1$ is called the *terminator*, because its occurrence results in the pattern being detected.

Second, sub-patterns once used are not used for detecting another instance of the same pattern, i.e., patterns need to be *unique*. For example, it does not make intuitive sense to combine $cell^2$ with $nucleic^2$, because $cell^2$ has already been used in a combination. Combining it again with $nucleic^2$ will result in the detection of another instance of the same pattern using a previously used sub-pattern. Of course, the above takes the distance into consideration where specified.

The Proximal-Unique semantics has been defined to take this intuitive sense of proximity and uniqueness into consideration when detecting a pattern by applying restrictions on the usage of sub-patterns. It is less complicated to detect patterns with unrestricted semantics although a large number of them are likely to be generated.

Non-overlap or disjoint aspect is also assumed. For example, suppose we want to find the occurrence of (“ $cell$ ” FOLLOWED BY “ $nucleic$ ”) NEAR (“ $protein$ ” FOLLOWED BY “ $clustering$ ”). According to the semantics discussed above, “ $cell$ ” FOLLOWED BY “ $nucleic$ ” occurs in the interval (15, 28) and “ $protein$ ” FOLLOWED BY “ $clustering$ ” occurs in the interval (20, 34). The sub-patterns satisfy the condition of being proximal, and of being the most recent un-combined occurrence of their type. However, they overlap and are not disjoint. Under the disjoint constraint, the combined pattern is not detected. It is also possible to relax the disjoint constraint in which case the NEAR operator will detect the above pattern.

2.3 Inverted Index

The inverted index (also called an inverted list) is the most common mechanism used in Search Engines [27] to maintain a mapping from a keyword to the documents that contain the keyword. Given a collection of documents, IDs are assigned to each document. A document ID uniquely identifies a document. The basic information stored in the inverted index is just a keyword - document ID mapping. For example, a sample set of documents is shown in Table 1 and the corresponding inverted index is shown in Table 2. This information is sufficient to answer simple keyword queries and queries involving Boolean operators. In other words, given a keyword, we can return document IDs of documents that contain at least one occurrence of that keyword. For example, in the given example, if the user is searching for “ $information$ ” AND “ $retrieval$ ”, the intersection of the document IDs corresponding to the keywords “ $information$ ” and “ $retrieval$ ” gives us the desired result (documents 1 and 3 in this example).

However, to answer queries involving proximity, sequences, frequency and containment, this information is *not* sufficient. First, the above scheme does not store information about *every* occurrence of a keyword. It only provides information about the presence or absence of a term within a document. Second, to answer such complex

Table 1. A sample set of documents

Document ID	Document contents
1	information retrieval
2	Specifying complex queries
3	information on information retrieval

Table 2. Inverted index on documents in Table 1

Keyword	Documents
information	1,3
retrieval	1,3
Specifying	2
complex	2
queries	2
on	3

Table 3. Inverted index with position information

Keyword	Documents with position
information	1<1>, 3<1,3>
retrieval	1<2>, 3<4>
Specifying	2<1>
complex	2<2>
queries	2<3>

queries, we need to compute the distance between two given patterns, and also the relative order of occurrence of these patterns. For example, a query such as “*information*” NEAR/2 “*retrieval*” cannot be answered using information from such an index, because the distance between occurrences of “*information*” and “*retrieval*” within a given document needs to be computed. This distance cannot be computed given just the document which the patterns belong to. The position of *every* occurrence of the keyword within a document must also be provided by the index [8]. Table 3 shows an inverted index generated on the documents in Table 1 with the position information stored.

Hence, InfoSearch needs at least the document ID and the position of a given keyword from the index with which it is integrated, in order to detect complex patterns. One of the main goals of this work was to assess whether this information is sufficient to enable complex pattern detection over an index, if the same patterns can be detected by reading the data source in sequence.

2.4 Pattern Detection Graphs

Patterns are detected using a data structure called Pattern Detection Graph (PDG). A query submitted to InfoSearch is converted into a PDG. Leaf nodes of the PDG correspond to simple patterns such as keywords, phrases or system defined patterns. Internal nodes correspond to complex patterns and encapsulate the logic of the corresponding operator. For example, the PDG corresponding to the pattern “*Protein*” FOLLOWED BY “*clustering*” is shown in Figure 2. The input to a leaf node is a set corresponding to the index lookup for the term or phrase represented by the leaf node. This set consists of $\langle docID, start\ offset, end\ offset \rangle$ tuples. As shown in the figure, “protein” occurs once at offset 10 in document 1 and “clustering” occurs once at offset 12 in document 1. As another example, the set of tuples for the keyword “*information*” from the index shown in Table 3 is: $1\langle 1,1 \rangle, 3\langle 1,1 \rangle$ and $3\langle 3,3 \rangle$.

Every node in a PDG has one or more parent nodes (also called as subscriber nodes), except the root node. Leaf nodes propagate their input sets to their parent nodes. A parent node, which corresponds to one of the operators such as OR or NEAR, thus gets one or more sets of tuples as its input. The operator merges its input sets according to the Proximal-Unique semantics for that operator to create an output set. After the merged set is created, it is propagated to the parent node of the operator. This process of propagating merged sets continues all the way up to the root. The merged output of the root operator corresponds to the result set for the query. For example, in figure 2, the input sets from leaf nodes are propagated to the “followed by” node, where the complex pattern is detected over the interval $\langle 10, 12 \rangle$.

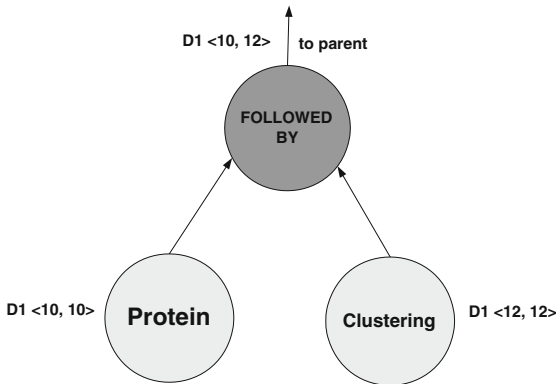


Fig. 2. PDG corresponding to “Protein” FOLLOWED BY “Clustering”

3 Pattern Operator Processing

InfoSearch computations are different from that of the algorithms used in a streaming system. In a streaming system [5], the operators work by reading the data source sequentially, and passing simple pattern occurrences to the respective PDG nodes as and when they occur while the data is being read. In other words, the input to a leaf node will be a tuple and not a set of tuples. Because the data is read sequentially, simple patterns are detected in their order of occurrence in the data source. As a result, at any operator, the initiator is always available when the terminator arrives. The occurrences can then be combined and propagated, or discarded, as per the semantics of the operator.

However, in InfoSearch the entire result set corresponding to a pattern is propagated at once because of the stored text. This means that the relative order of occurrence of the operands is lost, because each operand is a *set* containing all occurrences of the pattern corresponding to that operand in the document collection. Hence, to generate correct results, the InfoSearch operators need to restore the order of occurrence of patterns as in the original document. This is crucial in order to determine which operand is the initiator and which one is the terminator. Only when the relative order of occurrence

and position of sub-patterns is known, can a decision be made whether they can be combined or not.

The inputs to the operators are sets of tuples containing the document ID, start offset, and end offset of the corresponding pattern. Each tuple represents a single occurrence of the corresponding pattern in the document collection. It is assumed that these sets of tuples are sorted in ascending order of document ID and by start offset within each document ID. The operators have to process the input sets tuple by tuple. First, they have to ensure that the tuples to be merged have the same document ID. Second, they have to determine which tuple is the initiator and which one is the terminator. Tuples satisfying the criteria of the operator are combined and added to an output set. After the operator is done processing the input sets, the output set is propagated to its parent.

Due to space limitations, we discuss only the NEAR operator. Please refer to [9] for other operator and algorithm discussions.

3.1 The NEAR Operator

When the NEAR operator is processing two tuples from the input sets, it has to make a decision whether the tuples are eligible for combination, and if not, decide which one to keep and which one to discard. As mentioned earlier, the input tuples may either be point tuples or interval tuples. To keep the forthcoming discussion generalized, we assume that the input tuples have both start and end offsets. We now discuss the different cases possible when we consider two input tuples, and the corresponding actions taken in each case.

Merging strategy in NEAR: The inputs to the NEAR operator are two sets of tuples corresponding to the left child and the right child, and an optional distance. Let the left set be denoted by L and the right set by R . Let the distance be denoted by d . We arbitrarily assign the first tuple from L as *initiator*, and the first tuple from R as *terminator*. Let i_s and i_e denote the start offset and end offset of the *initiator*, and t_s and t_e denote the start offset and end offset of the *terminator*. Let $i + 1$ be the next tuple from the set which *initiator* belongs to, and $t + 1$ be the next tuple from the set which *terminator* belongs to.

If *initiator* and *terminator* do not belong to the same docID, we advance the pointer which is pointing to a smaller docID. Since the sets are sorted by docID, this is similar to a sort-merge operation. When *initiator* and *terminator* point to tuples belonging to the same docID, three cases are possible.

Case 1 ($i_e < t_e$): This means that the assumed *initiator* ends before the assumed *terminator*. The different possibilities are shown in Figure 3. We perform the following sequence of actions:

if *initiator* and *terminator* overlap **then**

lookahead² to determine new *initiator* and *terminator*

 go to the beginning of this operation and re-process the new *initiator* and *terminator*

² The Lookahead algorithm is explained below.

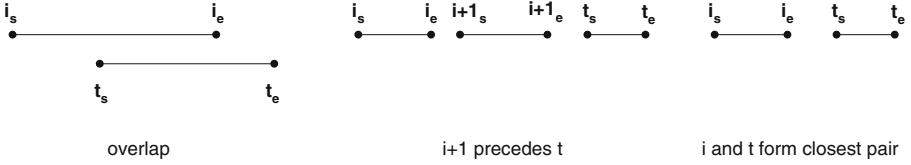


Fig. 3. Some possibilities when $i_e < t_e$

if $i + 1 \leq t_e$ **then**

make $i + 1$ the new *initiator*, and re-process the new *initiator* and *terminator*

else

this means *initiator* completely precedes *terminator*, without any overlap, and there is no other tuple from the *initiator* set occurring before *terminator*. Now, check if the distance criterion is satisfied.

if $(t_s - i_e) \leq d$ **then**

combine *initiator* and *terminator*

advance *initiator* and *terminator*

else

does not satisfy distance

lookahead to determine new *initiator* and *terminator*, re-process them

Case 2 ($i_e == t_e$): This means *initiator* and *terminator* overlap (they have the same end offset). Perform a lookahead, and re-process.

Case 3 ($i_e > t_e$): This means our assumption of *initiator* and *terminator* is wrong. The terminator precedes the initiator, either in an overlapping fashion, or a non-overlapping fashion as shown in Figure 4. In this case, we swap the *initiator* and *terminator* pointers, and re-process.

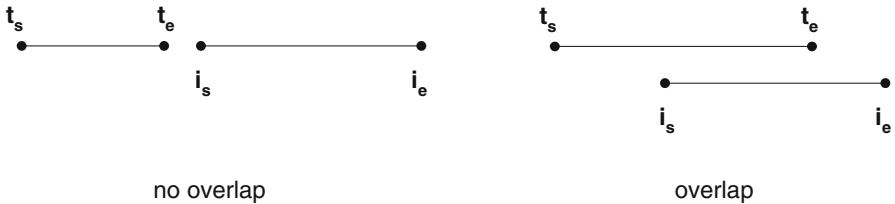


Fig. 4. Some possibilities when $t_e < i_e$

Lookahead algorithm: A lookahead is done when the current *initiator* and *terminator* cannot be combined due to an overlap, or because the distance criterion is not satisfied. At this point, we cannot determine which one from *initiator* and *terminator* to keep, and which one to discard. We look ahead one tuple from both sets, and assign the one that occurs first as the new terminator. The older tuple from the

opposite set becomes the new *initiator*. Three possibilities exist when we consider the lookahead tuples:

Case I ($i + 1_e < t + 1_e$): This means the next tuple in the *initiator* set occurs before the next tuple in the *terminator* set. (We assume they belong to the same docID).

Make old *terminator* the new *initiator*

Make $i + 1$ the new *terminator*

Case II ($i + 1_e == t + 1_e$): This means the next tuples have the same end offset. In this case, we look at the older pair, and keep the one that occurs later as the new *initiator*.

if $i_e < t_e$ **then**

Make old *terminator* the new *initiator*

Make $i + 1$ the new *terminator*

else

This means *initiator* and *terminator* have the same end offset

Keep the *initiator*

Make $t + 1$ the new *terminator*

Case III ($i + 1_e > t + 1_e$):

Keep the *initiator*

Make $t + 1$ the new *terminator*

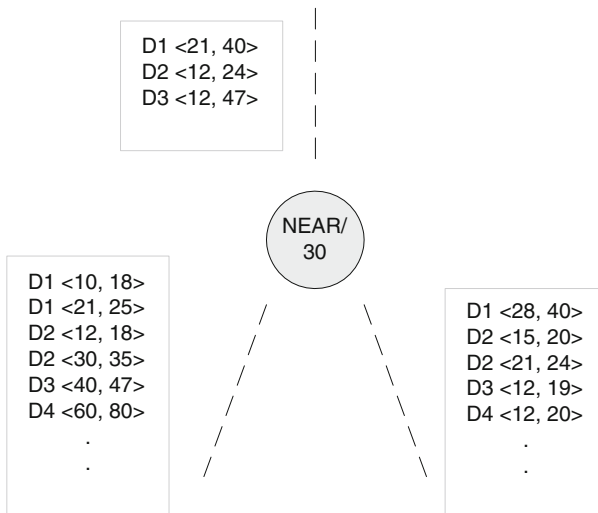


Fig. 5. Example of the NEAR operator algorithm

Figure 5 shows an example of the working of the NEAR operator. To begin, *initiator* points to D1 <10, 18> in the left set, and *terminator* points to D1 <28, 40> in the right set. Since the next tuple in the *initiator* set lies completely before *terminator*, it is assigned as the new *initiator* (*initiator* is advanced). Now, *initiator* and *terminator* point to a proximal pair of tuples, and hence they are merged and added to the output set as the tuple D1 <21, 40>. When *initiator* and *terminator* point to D2 <12, 18> and D2 <15,

20> respectively, an overlap is detected, and hence a lookahead is done in both sets. The lookahead determines that the next tuple from the right set ($D2 <21, 24>$) ends before the next tuple from the left set ($D2 <30, 35>$). Hence, $D2 <21, 24>$ is made the new *terminator* and $D2 <12, 18>$ is retained as the initiator. They are combined to form the output tuple $D2 <12, 24>$. Now, *initiator* points to a $D2$ tuple while *terminator* points to a $D3$ tuple. Hence, *initiator* is advanced. Now, *initiator* ($D3 <40, 47>$) lies completely after *terminator* ($D3 <12, 19>$). Hence, *initiator* and *terminator* are swapped. This makes *initiator* point to $D3 <12, 19>$ and *terminator* point to $D3 <40, 47>$, which form a proximal pair and are merged to give $D3 <12, 47>$ in the output set. Finally, *initiator* points to $D4 <12, 20>$, and *terminator* points to $D4 <60, 80>$. In this case, the distance between them is 40, which is greater than the maximum allowed distance, i.e., 30. Hence, they are not combined, and a lookahead needs to be done to determine which one of them should be discarded, and which one kept.

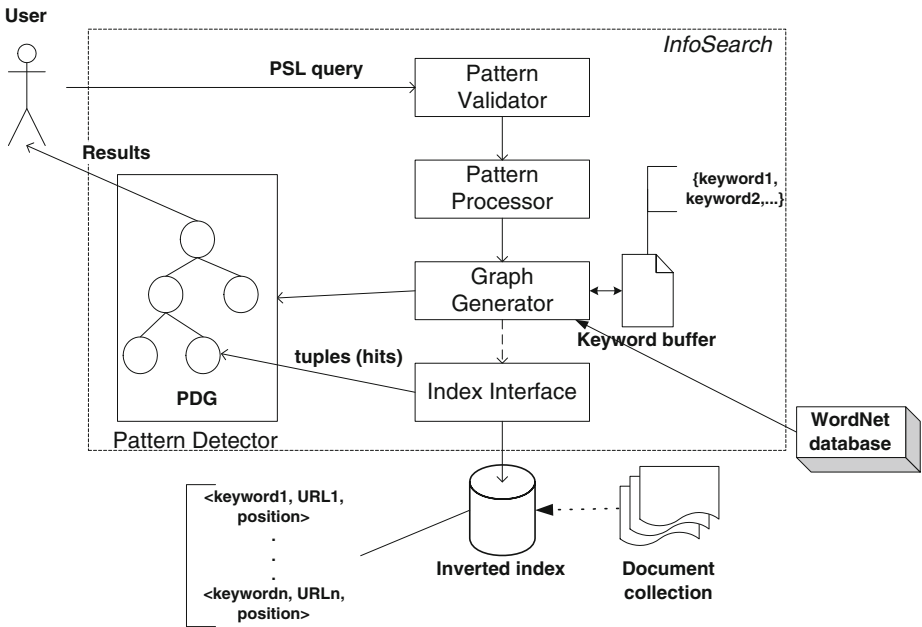


Fig. 6. InfoSearch architecture

4 Design and Implementation of InfoSearch

The InfoSearch architecture is shown in Figure 6. The user query specified in Pattern Specification Language is converted into a Pattern detection graph (or PDG). Leaf nodes of the PDG represent simple patterns such as keywords, phrases or system defined patterns. Higher level nodes represent composite operators on these leaf nodes, or on other composite nodes. To detect and optimize common computations, the *graph generator* shares PDG nodes (and sub-graphs) wherever possible. This is achieved by generating

a single, common PDG or sub-PDG for a common expression or sub-expression. While generating the graph, the *graph generator* stores the keywords specified in the query in a keyword buffer. Once the PDG is generated, the *graph generator* queries the index for each of the keywords it has stored in its buffer. This is done through the *index interface* module, which is responsible for retrieving the “hits” for each keyword from the index. The detection engine of InfoSearch is designed to be generic and capable of working with any kind of index. The “hits” are then wrapped into a set of <docID, start offset, end offset> tuples “tuples” and passed on to the leaf node that represents the keyword. Leaf nodes propagate their input to their parent nodes. The parent nodes, which correspond to one of the operators, merge their input sets according to the appropriate semantics.

4.1 Implementation

Whenever the *graph generator* comes across a token which is a keyword or a phrase, it stores this token in a *Vector* object called the **keyword buffer**. The keywords in the buffer are passed to the *index interface* after the PDG construction is complete, whereas the phrases are passed to the *phrase processor*. The reason for having a keyword buffer is that it is essential that the PDG is completely constructed before the index can be queried for the keywords. If the keywords are passed to the *index interface* or *phrase processor* by the *graph generator* as and when it pops them off the stack, they will return the results from the index to the PDG possibly before it is completely constructed. Thus, the keyword buffer is essential to avoid triggering of PDG nodes by the *index interface* while the PDG is being constructed. If the synonyms option is chosen for any keyword in the query, the *graph generator* queries the WordNet synonym database to get synonyms for the keyword. This is done through an API called the Java WordNet Library (JWNL) [10]. For each synonym, a leaf node is constructed, and finally a SYN operator node is constructed which subscribes to the original keyword and all its synonyms.

The *index interface* has to provide standard methods to access data from the integrated index, and deliver the results to the *pattern detector* in a specific format. As such, it does not matter if the index being integrated is an inverted index, or any other kind of index, say a B-tree index, as long as an index interface for it is developed. In other words, if a new index has to be integrated with InfoSearch, an *index interface* for that index has to be created which will support the required calls from InfoSearch, and return data to it in the expected format.

The *pattern detection engine* is responsible for processing the result sets from the index. The *index interface* passes a reference to a *Vector* of *Tuples* corresponding to a keyword to the leaf node corresponding to the keyword. Internal nodes of the PDG correspond to one of the operators. They get references to one or more *Vectors* from their children and merge them to produce an output *Vector*. This merging is done as per the operator semantics described earlier.

For the first release of this system, we built a simple inverted index using Berkeley DB Java Edition [11], and integrated it with InfoSearch. Since the Berkeley DB API is in Java, it was convenient to develop an *index interface* for it, because the rest of the InfoSearch system was also developed in Java. To create the inverted index, we

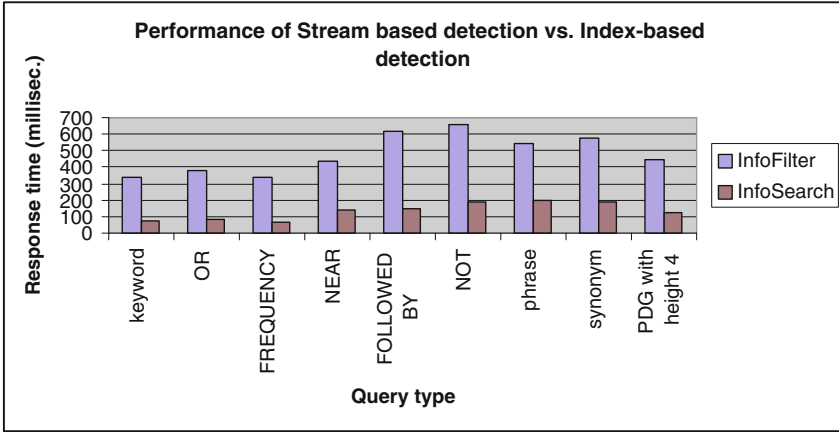


Fig. 7. Comparison of system performance over 2600 words

Operators \ Dataset Size	1MB		2MB		5MB		10MB		20MB		50MB	
	IS	IF	IS	IF	IS	IF	IS	IF	IS	IF	IS	IF
Keyword	5.33	6395	4.67	24651	4	18507	4	7779	3.33	7955	4.5	7456
OR	38	12249	92.5	24704	69	62832	68	50665	72.5	49271	73	119486
FREQUENCY	21.67	12379	18.67	23875	17	40575	18	34081	19.67	11173	19.5	32713
NEAR	23	10054	39	24184	53.67	62064	44.5	11970	39.33	30332	50	28767
FOLLOWED BY	21	12652	39.33	24528	63	61419	54	122912	54	260296	67.5	659407
NOT	50	12662	64.67	24347	93	62378	94	124080	96	250709	103.33	678606
PHRASE	32	12265	29.67	24417	63.5	62365	44.5	116913	43.5	248182	64.3	633933
SYN	40.67	12748	42	24169	54	15503	40.67	114311	42	113739	47	654118
PDG WITH HEIGHT 4	44.67	12849	47.33	15845	76.5	61585	82.5	124865	82	257901	105	845303

Fig. 8. Response Time of InfoSearch (IS) and InfoFilter (IF) Systems in milliseconds for each Operator

have created a Java program called *DocumentIndexer* which takes a given folder of documents, reads the documents, and builds an inverted index over those documents. For every keyword in each document, it stores a “hit” in the inverted index, which contains the path of the document the keyword is from, and the position of the keyword in that document.

5 Experimental Results

The primary reason for developing operators and algorithms to detect complex patterns over indexed data was to support efficient searching of stored documents for complex patterns. It does not make sense to stream *already stored* documents and use InfoFilter [5] for detecting patterns. Since InfoSearch uses an index-based approach, it is expected to be efficient for large volumes of data. A set of 20 documents of around 1.5 KB each were selected from the Reuters-21578 dataset [6] and the documents were artificially

³ Available at <http://www.daviddlewis.com/resources/testcollections/reuters21578>

converted into a stream, fed to InfoFilter [5] and patterns involving all the operators were detected over this document stream. The time taken to process the stream was noted in milliseconds. Subsequently, the same documents were indexed for testing the efficiency of InfoSearch and the time taken for the result set to reach the root node was noted in milliseconds. For each of the operators, the performance of these systems are noted and are shown in Figure 7. The experiments were again repeated with document sets of sizes 1MB, 2MB, 5MB, 10MB, 20MB and 50MB and the results are shown in Figure 8. The last row of the table indicates a pattern whose PDG is height 4 (has 4 operators and at least 5 leaf nodes). From that row it is clear that the improvement for data sizes as little as 1MB is more than a factor of 100. This is even better with the improvement being more than a factor of 1000 for data volumes of 50MB. As can be seen, the detection for index-based algorithm grows sub-linearly whereas the detection time for the streaming approach grows super-linearly. The time taken to index the documents is not considered in the above comparison (which is negligible as compared to the improvements in detection time), since it is a pre-processing step, and is amortized over multiple searches on the repository.

6 Related Work

Most search engines use a variation of the vector space model [1] to select documents against a query from a document collection. In addition, search engines try to add other factors to the ranking process for documents including external (meta) information about the documents, references to documents from other documents, etc. Google [2] stores the pages fetched by the crawler in compressed form in a repository. It has a document index, which is a fixed width ISAM index, to keep information about each document. It also has a lexicon, forward index and an inverted index to facilitate rapid access to document lists. However, it support queries only in the form of keywords and Boolean compositions of keywords. INQUERY [3] is based on a form of the probabilistic retrieval model called the inference net. Inference nets [4] provide the capability to specify complex information needs, and compare them to document representations. The operators supported by INQUERY include *and*, *or*, *not*, a phrase operator and also an operator that handles proximity between patterns. In addition, specification of a particular argument as being more important than the others can be done. *However, there are no operators for sequence of patterns, pattern frequency, synonyms and containment.*

7 Conclusions

It was observed that current search systems are somewhat restrictive in the expressiveness of patterns that can be specified by the user. InfoSearch facilitates searching of complex patterns involving proximity, frequency, containment and sequences over a given document collection. The use of pattern operators and its modified semantics to provide an expressive pattern specification mechanism and to develop algorithms for an index-based approach are the main contributions of the paper. We have demonstrated that there is no loss of detection capability from stream mode to index mode for the pattern specification language. The overhead of additional information is quite small

in the form of offsets which can be readily obtained while indexing. The index-based algorithms are quite different from their counterparts in stream processing.

We are currently working on incremental algorithms where the computation can be stopped after detecting k patterns efficiently without having to use and compute all patterns. With this it is also possible to consider ranking the results for their utility from a users' viewpoint.

References

1. Salton, G., Wong, A., Yang, C.: A vector space model for automatic indexing. *Communications of the ACM* 18, 613–620 (1975)
2. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. In: *Proc. of the WWW, Brisbane, Australia*, pp. 107–117 (April 1998)
3. Callan, J., Croft, B., Harding, S.: The inquiry retrieval system. In: *Proc. of the DEXA*, pp. 78–83 (1992)
4. Turtle, H., Croft, B.: Evaluation of an inference network-based retrieval model. *ACM Transactions on Information Systems* 9, 187–222 (1991)
5. Elkhalfi, L., Adaikkalavan, R., Chakravarthy, S.: Infofilter: A system for expressive pattern specification and detection over text streams. In: *Proc. of the ACM SAC, Santa Fe, NM (March 13-17, 2005)*
6. Chakravarthy, S., Elkhalfi, L., Deshpande, N., Adaikkalavan, R., Liuzzi, R.A.: How To Search for Complex Patterns Over Streaming and Stored Data. In: *IC-AI*, pp. 17–22 (2006)
7. Mauldin, M.L.: Lycos : Design choices in an internet search service. *IEEE Expert* (1997), <http://lazytoad.com/liti/pub/ieee97.html>
8. Witten, I., Moffat, A., Bell, T.: *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufman, San Francisco (1999)
9. Deshpande, N.: *Infosearch : A system for searching and retrieving documents using complex queries*, Master's thesis, University of Texas at Arlington, Arlington (2005), <http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Des05MS.pdf>
10. Java wordnet library, <http://sourceforge.net/projects/jwordnet>
11. Berkeley db java edition, <http://www.oracle.com/us/products/database/berkeley-db/je/index.html>

Using the Euclidean Distance for Retrieval Evaluation

Shengli Wu¹, Yaxin Bi¹, and Xiaoqin Zeng²

¹ School of Computing and Mathematics,
University of Ulster, Northern Ireland, UK
{s.wu1,y.bi}@ulster.ac.uk

² College of Computer and Information Engineering
Hehai University, Nanjing, China
xzeng@hhu.edu.cn

Abstract. In information retrieval systems and digital libraries, retrieval result evaluation is a very important aspect. Up to now, almost all commonly used metrics such as average precision and recall level precision are ranking based metrics. In this work, we investigate if it is a good option to use a score based method, the Euclidean distance, for retrieval evaluation. Two variations of it are discussed: one uses the linear model to estimate the relation between rank and relevance in resultant lists, and the other uses a more sophisticated cubic regression model for this. Our experiments with two groups of submitted results to TREC demonstrate that the introduced new metrics have strong correlation with ranking based metrics when we consider the average of all 50 queries. On the other hand, our experiments also show that one of the variations (the linear model) has better overall quality than all those ranking based metrics involved. Another surprising finding is that a commonly used metric, average precision, may not be as good as previously thought.

1 The Euclidean Distance

In information retrieval, how to evaluate results is an important problem. A lot of effort has been taken on this and some related issues. Many metrics for retrieval effectiveness have been proposed. Average precision (AP), recall level precision (RP), normalized discount cumulative gain (NDCG) [4], and average precision at 10 document level (P10) are four of the most commonly used metrics. One major characteristic of these metrics is: they only concern the ranking positions of relevant/irrelevant documents. They are referred to as ranking based metrics later in this paper.

In fact, apart from a ranked list of documents, some information retrieval systems also provide relevance scores for all retrieved documents. For example, for all those submitted runs to TREC [1], most of them provide such score information. Suppose for a collection D of documents $\{d_1, d_2, \dots, d_n\}$ and a given

¹ TREC stands for Text REtrieval Conference. It is an annual information retrieval evaluation event held by the National Institute of Standards and Technology of the USA. Its web site is located at <http://trec.nist.gov/>.

query Q , a result R from an information retrieval system IR is $\{s_1, s_2, \dots, s_n\}$, here s_i is the score assigned to document d_i by IR . On the other hand, for every document d_i , it has an ideal relevance score o_i . If binary relevance judgment is used, then the ideal score for any judged relevant document is 1 and that for any judged irrelevant document is 0. If 3 graded relevance judgment is used, then 0, 0.5, and 1 can be used as ideal scores of irrelevant, modestly relevant, and highly relevant documents, respectively. In such a situation, the Euclidean distance between the scores in R and the ideal scores $O = \{o_1, o_2, \dots, o_n\}$ can be calculated by

$$distance(R, O) = \sqrt{\sum_{i=1}^n (s_i - o_i)^2} \quad (1)$$

$distance(R, O)$ can be used as a metric to evaluate the effectiveness of R . If all documents' scores are estimated accurately, then we can expect a low Euclidean distance value; otherwise, we can expect a high one. The Euclidean distance has been widely used as a metric in many areas such as data mining, neural networks, etc. However, to our knowledge, it has never been explored in information retrieval. Different from those ranking based metrics, the Euclidean distance can be regarded as a relevance score based metric. It is an interesting thing to find out if the Euclidean distance is a good choice for the evaluation of information retrieval results.

At first glance, one may think that the condition for using this metric is very rigorous since relevance scores for all the documents in the whole collection need to be provided. This is only true theoretically. In practice, we may use some reasonable approximation methods. In TREC, only a certain number (say, 1000) of documents are included in a result for a query. It is not known what the scores are for those documents that do not occur in the result. Since those documents are very likely irrelevant, we can reasonably assign a default score of 0 to all of them. Another situation is: although some retrieval systems provide scores for all retrieved documents, those scores may be in various ranges (say, one is in the range of 1 to 1000 and another in the range of 0 to 10) and cannot be used directly as relevance scores. In such a situation, we may use score normalization methods to normalize all the scores to the desired range $[0,1]$. Several such methods have been investigated before [5,6,11]. Finally, some retrieval systems may not provide scores at all for the retrieved documents. Then we may assign different scores to documents at different ranks in a systematic manner. One straightforward method for this is the linear model: for a ranked list of m documents, the top ranked document is given a score of 1, the 2nd document in the list is assigned a score of $(m-1)/m, \dots$, the last document in the list is assigned a score of $1/m$. This is the method used in Borda voting. It has been used for data fusion [1] in information retrieval as well. Alternatively, we may use nonlinear models to estimate scores for documents at different ranks. For example, we may use the cubic model [9] or other models (e.g., the logistic model [3]) to do this. The cubic model can be expressed by the following equation

$$s(i) = a_0 + a_1 * \ln(i) + a_2 * \ln(i)^2 + a_3 * \ln(i)^3 \quad (2)$$

In Equation 2, $s(i)$ is the relevance score of the document at rank i . a_0 , a_1 , a_2 , and a_3 are 4 parameters.

Therefore, for any result, we can always evaluate it using the Euclidean distance after appropriate pre-processing. Some variations of the Euclidean distance can also be defined, see [10] for detailed discussion.

2 Investigation Objectives and Methodologies

The aims of the study are twofold: one is to evaluate the Euclidean distance, which is introduced in this paper; the other is to evaluate those ranking based metrics in the environment that 3 graded relevance judgment is used. There have been quite a few empirical investigations for those metrics when binary relevance judgment is used (e.g., in [2][7][8][13]). However, up to now very little has been done for them when relevant judgment methods other than binary relevance judgment are used.

Apart from the Euclidean distance, we also consider 4 ranking based metrics: AP, RP, NDCG, and P10, because they are four of the most commonly used metrics for retrieval evaluation. Making a comparison of these two types of metrics is helpful for us to have a better understanding of the characteristics of them.

For readers' convenience, we discuss how these metrics are defined. First let us see how to define these metrics involved when binary relevance judgement is used. Suppose for a query Q , an information retrieval system returns a list of documents R . There are $total_r$ relevant documents in the whole collection. AP is defined as

$$AP = \frac{1}{total_r} \sum_{i=1}^{total_r} \frac{i}{p_i}$$

Here p_i is the ranking position of the i -th relevant documents in the resultant list R . One thing needs to be noticed is: usually a very small percentage of documents in the whole collection are retrieved and included in any result, thus it is very likely that less than $total_r$ relevant documents will appear in such a result. Then we just assume that those missing relevant documents will never appear and their contribution to the value of AP is ignored. For example, if t relevant documents appear in R , then AP can be defined as

$$AP = \frac{1}{total_r} \sum_{i=1}^t \frac{i}{p_i}$$

For example, if there are 4 relevant documents in the whole collection and 2 of them are retrieved in the ranking positions of 2 and 4 in R , then $AP = 1/4 * (1/2 + 2/10) = 0.175$.

RP is defined as the percentage of relevant documents in the top $total_r$ documents in R .

P10 is defined as the percentage of relevant documents in the top 10 documents in R .

NDCG is introduced in [4]. Each ranking position in a resultant document list is assigned a given weight. The top ranked documents are assigned the highest weights since they are the most convenient ones for users to read. A logarithmic function-based weighting schema was proposed in [4], which needs to take a particular whole number c . The first c documents are assigned a weight of 1; then for any document ranked i which is greater than c , its weight is $w(i) = \ln(c)/\ln(i)$. Considering a resultant document list up to t documents, its discount cumulated gain (DCG) is defined as

$$DCG = \sum_{i=1}^t (w(i) * r(i))$$

if the i -th document is relevant, then $r(i) = 1$; if the i -th document is irrelevant, then $r(i) = 0$. DCG can be normalized using a normalization coefficient DCG_{best} , which is the DCG value of the best resultant lists. Therefore, we have:

$$NDCG = \frac{1}{DCG_{best}} \sum_{i=1}^t (w(i) * r(i))$$

Now let us see a way of extending AP, RP, and P10 for graded relevance judgement [12]. Note that NDCG can be used in the condition of graded relevance judgement directly, so no extension is needed for it.

Suppose there are n relevance grades ranging from 1 to n (n means the most relevant state and 0 means the irrelevant state), then each document d_i can be assigned a grade $g(d_i)$ according to its degree of relevance to the given query. One primary assumption taken for these documents in various grades is: a document in grade n is regarded as 100% relevant and 100% useful to users, and a document in grade i ($i < n$) is regarded as $i/n\%$ relevant and $i/n\%$ useful to users. Suppose there are $total_n$ documents whose grades are above 0 and $total_n = |r_1| + |r_2| + \dots + |r_n|$. Here $|r_i|$ denotes the number of documents in grade i . First let us see the concept of the best resultant list. For the given query Q , a resultant list L is best if it satisfies the following two conditions:

- * all the documents whose grades are above 0 appear in the list;
- * for any document pair d_i and d_j , if d_i is ranked in front of d_j , then $g(d_i) \geq g(d_j)$.

Many resultant lists can be the best at the same time, since more than one document can be in the same grade and the documents in the same grade can be arranged in different orders, but the relative ranking positions of documents in different grades cannot be changed. Therefore, we can use $g_best(d_j)$ to refer to the grade of the document in ranking position j in one of these best resultant lists. We may also sum up the grades of the documents in top $|r_n|$, top $(|r_n| + |r_n - 1|), \dots$, top $(|r_n| + |r_n - 1| + \dots + |r_1|)$ for any of the best resultant lists (these sums are the same for all the best resultant lists):

$$\begin{aligned}
b_n &= \sum_{i=1}^{|r_n|} g(d_i), \\
b_{n-1} &= \sum_{i=1}^{|r_n+r_{n-1}|} g(d_i) \dots, \\
b = b_1 &= \sum_{i=1}^{|r_n+r_{n-1}+\dots+r_1|} g(d_i)
\end{aligned}$$

AP can be defined as

$$AP = \frac{1}{b} \sum_{i=1}^{total_n} g(d_{p_i}) \sum_{j=1}^i \frac{g(d_{p_j})}{p_i}$$

Here p_j is the ranking position of the j -th document whose grade is above 0, and $\sum_{j=1}^i g(d_{p_j})$ is the total sum of grades for documents up to rank p_i . Considering all these $total_n$ documents in the whole collection whose grades are above 0, AP needs to calculate the precision at all these document levels ($p_1, p_2, \dots, p_{total_n}$). At any p_i , precision is calculated as $\sum_{j=1}^i g(d_{p_j})/p_i$, and then a weight of $g(d_{p_i})$ is applied. In this way the documents in higher grades have a bigger contribution to the final value of AP.

For RP, First we only consider the top $|r_n|$ documents, $\frac{1}{b_n} \sum_{j=1}^{|r_n|} g(d_j)$ can be used to evaluate their precision; next we consider the top $|r_n| + |r_{n-1}|$ documents, $\frac{1}{b_{n-1}} \sum_{j=1}^{|r_n|+|r_{n-1}|} g(d_j)$ can be used to evaluate their precision, continue this process until finally we consider all top $total_n$ documents using $\frac{1}{b} \sum_{j=1}^{|r_n|+\dots+|r_1|} g(d_j)$. Combining all these, we have

$$RP = \frac{1}{n} \left\{ \frac{1}{b_n} \sum_{j=1}^{|r_n|} g(d_j) + \frac{1}{b_{n-1}} \sum_{j=1}^{|r_n|+|r_{n-1}|} g(d_j) + \dots + \frac{1}{b_1} \sum_{j=1}^{|r_n|+|r_{n-1}|+\dots+|r_1|} g(d_j) \right\}$$

With binary relevance judgment or graded relevance judgement, all these defined metrics are in the range of $[0,1]$. 0 is used to represent the least effective result and 1 is used to represent the most effective result.

For the investigation, we use two groups of runs submitted to TREC: 9 and 2001 Web tracks. One major reason for choosing these two groups is because three category relevance judgement is used for both; while in many other groups, binary relevance judgement is commonly used. From all 105 runs submitted to the TREC 9 Web track and 97 runs submitted to the TREC 2001 Web track, we select those that include 1000 documents for each of the queries. Thus we obtained 53 in TREC 9 and 34 runs in TREC 2001². Removing those runs with fewer documents provides us a homogeneous environment for the investigation, and it should be helpful for us to obtain more reliable experimental results.

² See the appendix for the list of runs selected in each group.

NDCG needs to set a parameter c , which we set to 2 as in [4]. As to AP, RP, and P10, we use their expanded form [12] described above for 3 graded relevance judgment. Two variations of the Euclidean distances were involved. They are the linear model and the cubic model for the estimation of relation between rank and relevance in resultant lists. Note that the raw scores of retrieved documents from information retrieval systems are not used in both variations. From this perspective, they are somewhat like ranking based metrics. This is the interesting part.

We applied the cubic model to all the selected runs in each year group and obtained the values of 4 parameters by regression analysis: For TREC 9, the values of the four parameters are: $a_0=0.2474$, $a_1=-0.0639$, $a_2=0.0036$, and $a_3=0.0001$. For TREC 2001, the values of the four parameters are: $a_0=0.3267$, $a_1=-0.0761$, $a_2=0.0032$, and $a_3=0.0002$. Thus, we can assign proper relevance scores to documents at different ranks. However, note that those parameter values are only reasonably good for the estimation of rank-relevance of each individual result. We avoided using the best possible parameter values for every individual result, so as to be fair to the other metrics.

3 Experiments

A few different aspects of these metrics are compared through three groups of experiments. Let us discuss them one by one.

3.1 Experiment 1

First, for all the selected runs in a year group (TREC 9 or TREC 2001), we evaluated the effectiveness of them over 50 queries using all 6 metrics. Then Pearson's correlation coefficients were calculated for the different rankings of the information retrieval systems obtained by using different metrics. The correlation coefficients are shown in Tables 1 and 2, for TREC 9 and TREC 2001, respectively.

In all the cases, the correlations are significant at the 0.01 level (2-tailed). Tables 1 and 2 shows that, generally speaking, there is a strong correlation between any of the two variations of the Euclidean distance and any of the four ranking based metrics. However, the strength of correlation varies across the two year groups and the two variations of the Euclidean distance. The smallest is .624 (between ED(L) and P10 in TREC 9) and the biggest is .981 (between ED(C) and NDCG in TREC 2001).

We also carried out the linear regression analysis for those values of different metrics. Tables 3-6 shows the coefficients and significance of the analysis. The Euclidean distance can be well or reasonably well expressed linearly using any of the four metrics. Among them, NDCG is always the best ($R^2=0.946$, 0.664, 0.799, and 0.962; if $R^2 = x$, then it means that NDCG can explain $x\%$ of the variation in the Euclidean distance and vice versa) and P10 ($R^2=0.782$, 0.389, 0.484, and 0.754) is always the least able to express the Euclidean distance, while AP and RP are in the second and third places, respectively.

Table 1. Pearson’s correlation coefficients for rankings of retrieval systems in TREC 9 by different metrics (ED(L): Euclidean Distance with the linear model; ED(C): Euclidean Distance with the cubic model)

Metric	AP	RP	NDCG	P10
ED(L)	.928	.905	.973	.884
ED(C)	.702	.663	.815	.624
AP		.992	.976	.962
RP			.969	.973
NDCG				.941

Table 2. Pearson’s correlation coefficients for rankings of retrieval systems in TREC 2001 by different metrics (ED(L): Euclidean Distance with the linear model; ED(C): Euclidean Distance with the cubic model)

Metric	AP	RP	NDCG	P10
ED(L)	.872	.799	.894	.696
ED(C)	.952	.932	.981	.869
AP		.952	.975	.872
RP			.955	.918
NDCG				.897

Table 3. Linear regression of different metric values in TREC 9 (dependent variable is ED(L))

Metric	Constant	Linear coefficient	R^2	Significance level
AP	18.597	-3.313	0.862	.000
RP	18.616	-2.785	0.820	.000
NDCG	18.724	-1.771	0.946	.000
P10	18.593	-3.377	0.782	.000

3.2 Experiment 2

Next we carried out an experiment to compare the overall quality of all metrics. In [28], the stability of several metrics was investigated. Here we took a slightly different approach, which we think is more reliable. Using a certain metric, we compare the average effectiveness of two runs A and B over 50 queries to see if the difference between them is above a given threshold (T). If it is true, or $e(A) > e(B)$ and $(e(A) - e(B))/e(A) > T$, then we look at every query to see how many of them will contradict the above conclusion, or $(e(B) - e(A))/e(B) > T$. Thus error rate can be calculated as the percentage of queries which are contradicted to the overall conclusion. On the other hand, for a given threshold (T), we calculate how many pairs of runs can be distinguished from all possible pairs. The percentage of these (differentiation rates) is used to represent the sensitivity of a metric. Since the Euclidean distance and all ranking based measures are very different, It is not a good idea to use the same thresholds for them. Instead, we

Table 4. Linear regression of different metric values in TREC 9 (dependent variable is ED(C))

Metric	Constant	Linear coefficient	R^2	Significance level
AP	9.013	-.675	0.493	.000
RP	9.013	-.548	0.439	.000
NDCG	8.972	-.399	0.664	.000
P10	9.021	-.641	0.389	.000

Table 5. Linear regression of different metric values in TREC 2001 (dependent variable is ED(L))

Metric	Constant	Linear coefficient	R^2	Significance level
AP	18.412	-2.682	0.760	.000
RP	17.749	-2.403	0.639	.000
NDCG	18.736	-1.840	0.799	.000
P10	18.243	-1.500	0.484	.000

Table 6. Linear regression of different metric values in TREC 2001 (dependent variable is ED(C))

Metric	Constant	Linear coefficient	R^2	Significance level
AP	4.688	-1.227	0.906	.000
RP	4.726	-1.173	0.868	.000
NDCG	4.838	-0.845	0.962	.000
P10	4.633	-0.784	0.754	.000

used different thresholds for them so as to let the differentiation rate be in the range that we are interested. For the Euclidean distance, we used 10 thresholds (0.1%, 0.2%, ..., 1%); while for ranking based metrics, we used 10 thresholds (6%, 9%, ..., 33%). In TREC 9, there are 53 runs and the number of all possible pairs of runs is 1431. In TREC 2001, there are 34 runs and the number of all possible pairs of runs is 561. Let us take TREC 2001 as an example. Assuming for a given threshold T and a given metric there are n pairs of runs whose performance difference is above T from all possible 561 pairs. In this situation, the differentiation rate of the metric m is $n/561$, for the given threshold T .

Tables 7-10 show the experimental results. It is obvious that a good metric should have high differentiation rates and low error rates at the same time, though these two aspects are somewhat conflicting. Therefore, we define the "overall quality" of a metric as D_rate/E_rate for any given threshold. Here D_rate is the differentiation rate and E_rate is the error rate.

Let us compare the two variations of the Euclidean distance and then the four ranking based metrics separately. From the angle of differentiation rate, the cubic model performed better than the linear model in TREC 9 but worse than the linear model in TREC 2001; as for error rate, the cubic model performed not

Table 7. Evaluation of the Euclidean distance (TREC 9, 1431 pairs in total; Overall quality(Q)=D_rate/R_rate; T: Threshold)

T	ED(L)			ED(C)		
	D_rate	E_rate	Q	D_rate	E_rate	Q
0.1%	89.2%	17.0%	5.2	82.8%	29.4%	2.8
0.2%	83.0%	13.1%	6.3	71.7%	23.9%	3.0
0.3%	78.8%	10.7%	7.4	62.0%	18.9%	3.3
0.4%	74.5%	8.6%	8.7	52.8%	15.2%	3.5
0.5%	70.7%	7.2%	9.9	44.9%	11.7%	3.8
0.6%	65.9%	5.9%	11.1	38.9%	8.5%	4.6
0.7%	61.7%	4.9%	12.6	34.1%	6.1%	5.6
0.8%	57.9%	4.0%	14.5	29.0%	4.6%	6.4
0.9%	52.9%	3.0%	17.7	26.8%	3.6%	7.5
1.0%	49.6%	2.5%	19.9	25.4%	1.8%	13.4

Table 8. Evaluation of the Euclidean distance (TREC 9, 1431 pairs in total; Overall quality(Q)=D_rate/R_rate; T: Threshold)

T	AP			RP			NDCG			P10		
	D_rate	E_rate	Q	D_rate	E_rate	Q	D_rate	E_rate	Q	D_rate	E_rate	Q
6%	82.5%	22.7%	3.6	83.9%	18.9%	4.4	80.2%	19.0%	4.2	81.5%	17.5%	4.7
9%	75.3%	20.2%	3.7	76.9%	17.0%	4.5	71.1%	15.4%	4.6	74.6%	16.1%	4.6
12%	70.2%	18.2%	3.9	69.8%	15.0%	4.6	64.3%	12.7%	5.1	69.3%	14.8%	4.7
15%	64.1%	16.0%	4.0	63.8%	13.0%	4.9	57.8%	10.5%	5.5	65.7%	13.6%	4.6
18%	59.2%	14.3%	4.1	58.7%	11.6%	5.1	51.9%	8.7%	6.0	61.9%	12.6%	4.9
21%	54.6%	12.7%	4.3	53.2%	10.3%	5.2	46.4%	6.9%	6.8	59.6%	11.7%	5.1
24%	50.2%	11.3%	4.4	49.8%	9.4%	5.3	42.1%	5.9%	7.1	57.1%	11.2%	5.1
27%	50.0%	10.2%	4.6	46.3%	8.2%	5.6	39.2%	4.9%	8.1	53.7%	9.9%	5.4
30%	45.0%	9.5%	4.7	43.9%	7.6%	5.8	35.8%	3.9%	9.2	49.6%	9.1%	5.4
33%	42.8%	8.9%	4.8	41.7%	7.2%	5.8	32.9%	3.3%	10.1	46.0%	8.6%	5.4

Table 9. Evaluation of the Euclidean distance (TREC 2001, 561 pairs in total; Overall quality(Q)=D_rate/R_rate; T: Threshold)

T	ED(L)			ED(C)		
	D_rate	E_rate	Q	D_rate	E_rate	Q
0.1%	88.9%	30.5%	2.9	92.3%	36.3%	2.5
0.2%	80.0%	26.2%	3.1	87.0%	34.5%	2.5
0.3%	68.1%	22.9%	2.9	81.5%	32.7%	2.5
0.4%	59.9%	20.2%	3.0	75.9%	31.2%	2.4
0.5%	51.2%	17.7%	2.9	71.3%	29.6%	2.4
0.6%	44.9%	15.8%	2.8	66.1%	27.8%	2.4
0.7%	40.3%	14.5%	2.8	59.9%	26.0%	2.3
0.8%	34.8%	13.2%	2.6	52.9%	24.2%	2.2
0.9%	29.8%	11.7%	2.6	47.1%	22.7%	2.1
1.0%	25.0%	10.2%	2.5	43.5%	21.4%	2.0

Table 10. Evaluation of the Euclidean distance (TREC 2001, 561 pairs in total; Overall quality(Q)=D_rate/R_rate; T: Threshold)

T	AP			RP			NDCG			P10		
	D_rate	E_rate	Q	D_rate	E_rate	Q	D_rate	E_rate	Q	D_rate	E_rate	Q
6%	78.4%	33.0%	2.4	75.4%	29.0%	2.6	66.1%	28.1%	2.4	73.3%	25.1%	2.9
9%	67.0%	30.2%	2.2	59.4%	25.6%	2.3	50.1%	23.1%	2.2	64.3%	23.4%	2.7
12%	56.1%	27.9%	2.0	51.1%	23.0%	2.2	36.5%	18.4%	2.0	56.0%	21.4%	2.6
15%	46.7%	24.9%	1.9	42.8%	20.5%	2.1	27.3%	14.3%	1.9	49.4%	18.7%	2.6
18%	37.8%	21.9%	1.7	33.2%	17.7%	1.9	20.3%	11.3%	1.8	43.5%	17.3%	2.5
21%	30.1%	19.1%	1.6	26.6%	15.4%	1.7	14.8%	8.7%	1.7	38.5%	15.3%	2.5
24%	23.9%	16.8%	1.4	20.7%	13.5%	1.5	10.7%	7.8%	1.4	35.1%	14.5%	2.4
27%	20.0%	15.6%	1.3	14.8%	12.7%	1.2	8.7%	7.8%	1.1	29.1%	13.1%	2.2
30%	16.0%	14.1%	1.1	11.2%	11.6%	1.0	6.0%	7.2%	0.8	23.5%	12.2%	1.9
33%	12.1%	12.3%	1.0	8.2%	11.5%	0.7	4.5%	6.5%	0.7	18.9%	11.1%	1.7

as good as the cubic model in both year groups. Therefore, the “overall quality” of the cubic model is not as good as the linear model.

Now let us look at the four ranking based metrics. For differentiation rate, P10 performed best and AP performed second to the best in both year groups. However, for error rate, AP was the worst in both year groups (see Figure 1). That is why its “overall performance ” is the worst among the four metrics. On the other hand, NDCG has the lowest differentiation rate and error rate in both year groups, and has the best “overall quality”. AP is in the second place, and very close to NDCG in “overall quality”.

If we consider the “overall quality” of all the metrics, then the averages are: ED(L): 7.1; NDCG: 4.1; ED(C): 3.8; P10: 3.7; RP; 3.4; AP: 2.9. The Euclidean distance with the linear model is the best, which has a much higher average score than all the others. It is quite surprising to see that average precision (AP) is the worst, with an average score of 2.9. In the information retrieval community, AP is commonly used and regarded as a very good system-oriented metric for retrieval evaluation [28].

3.3 Experiment 3

The above experiment should be reliable for the comparison of the two variations of the Euclidean distance, or the comparison of all those ranking based metrics separately. However, since the threshold settings of performance difference are different for the Euclidean distance and ranking based metrics, the conclusions may not be very convincing for the comparison of the Euclidean distance and ranking based metrics. In order to evaluate all these metrics in a more comparable style, we carried out another experiment. This time we set up a fixed group of differentiation rates (0.2, 0.25, ..., 0.6), and then find the corresponding threshold for each of them. Note that the differentiation rate decreases monotonously when the threshold increases. Using a threshold as such for a given differentiation rate, we find its corresponding error rate. Figures 2-3 shows the results.

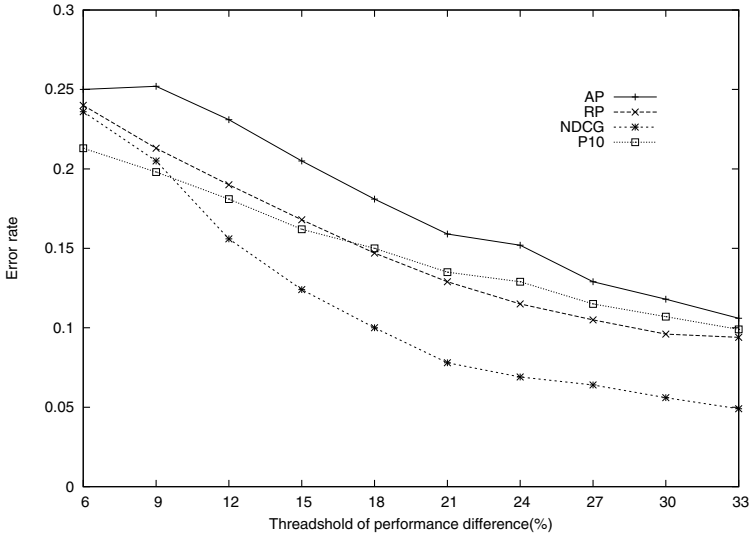


Fig. 1. Average error rates of four ranking based metrics when a given performance difference threshold is used

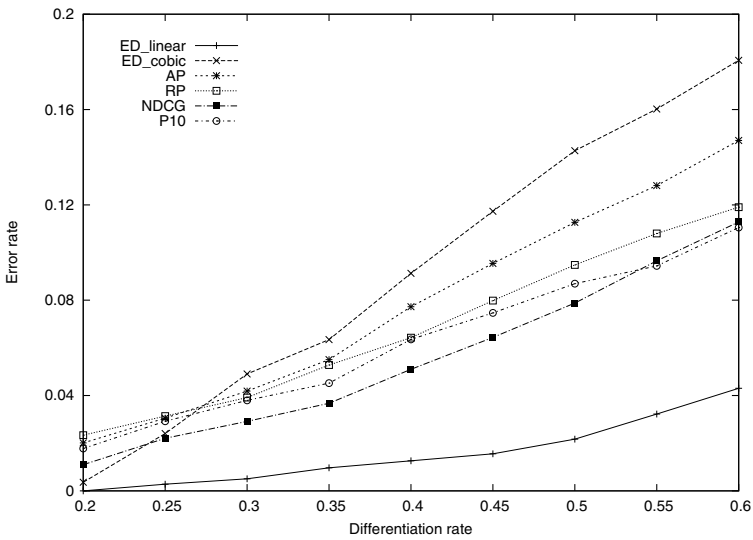


Fig. 2. Average error rates of the TREC 9 group in the condition of fixed differentiation rates

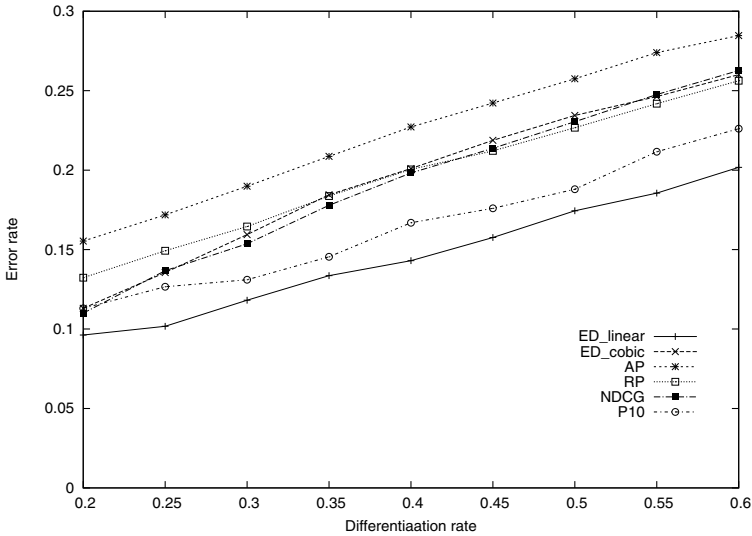


Fig. 3. Average error rates of the TREC 2001 in the condition of fixed differentiation rates

Form this experiment, we confirm that the Euclidean distance with the linear model is the best among all the metrics involved. However, this time the Euclidean distance with the cubic model becomes the worst. Those ranking based metrics are close. However, in TREC 9, NDCG is better than the three others; in TREC 2001, P10 is better than the three others. If considering the average of the two year groups, the order from best to worst is P10, NDCG, RP, and AP. It is exactly the same as that in the above experiment.

4 Conclusions

From our experiments, we demonstrate that the Euclidean distance with the linear model is a very good metric. It has the ability of keeping good balance between sensitivity and reliability. On the other hand, its cousin, the Euclidean distance with the cubic model, does not seem to be as good as it. It suggests that the Euclidean distance can be a good metric if relevance scores are properly estimated. In addition, the linear model is easy to apply and does not need any training data as the cubic model. As a matter of fact, it can be used in exactly the same way as those ranking based metrics. We believe it is an attractive option for retrieval evaluation. However, for the Euclidean distance, there is one problem: though the cubic model is a more sophisticated model than the linear model, why the linear model is better than the cubic model? This is not clear and remains to be our future investigation problem.

Another finding is that average precision may not be as good as people thought before, though our experimental results in this paper do not necessarily conflict with that from previous research due to two reasons as follows: first, the methodologies we have taken are slightly different from those in previous research. Second, the four metrics AP, RP, NDCG, and P10 used in our investigation are the expanded forms for graded relevance judgment, while in previous research their original form was used with binary relevance judgment. However, the experimental results reported in this paper provide some new evidence for the evaluation and comparison of these commonly used metrics.

References

1. Aslam, J.A., Montague, M.: Models for metasearch. In: Proceedings of the 24th Annual International ACM SIGIR Conference, New Orleans, Louisiana, USA, pp. 276–284 (September 2001)
2. Buckley, C., Voorhees, E.M.: Evaluating evaluation measure stability. In: Proceedings of ACM SIGIR Conference, Athens, Greece, pp. 33–40 (July 2000)
3. Calvé, A.L., Savoy, J.: Database merging strategy based on logistic regression. *Information Processing & Management* 36(3), 341–359 (2000)
4. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems* 20(4), 442–446 (2002)
5. Lee, J.H.: Analysis of multiple evidence combination. In: Proceedings of the 20th Annual International ACM SIGIR Conference, Philadelphia, Pennsylvania, USA, pp. 267–275 (July 1997)
6. Montague, M., Aslam, J.A.: Relevance score normalization for metasearch. In: Proceedings of ACM CIKM Conference, Berkeley, USA, pp. 427–433 (November 2001)
7. Sakai, T.: Evaluating evaluation metrics based on the bootstrap. In: Proceedings of ACM SIGIR Conference, Seattle, USA, pp. 525–532 (August 2006)
8. Sanderson, M., Zobel, J.: Information retrieval system evaluation: Effort, sensitivity, and reliability. In: Proceedings of ACM SIGIR Conference, Salvador, Brazil, pp. 162–169 (August 2005)
9. Wu, S., Bi, Y., McClean, S.: Regression relevance models for data fusion. In: Proceedings of the 18th International Workshop on Database and Expert Systems Applications, Regensburg, Germany, pp. 264–268 (September 2007)
10. Wu, S., Bi, Y., Zeng, X.: Retrieval result presentation and evaluation. In: Bi, Y., Williams, M.-A. (eds.) KSEM 2010. LNCS, vol. 6291, pp. 125–136. Springer, Heidelberg (2010)
11. Wu, S., Crestani, F., Bi, Y.: Evaluating score normalization methods in data fusion. In: Ng, H.T., Leong, M.-K., Kan, M.-Y., Ji, D. (eds.) AIRS 2006. LNCS, vol. 4182, pp. 642–648. Springer, Heidelberg (2006)
12. Wu, S., McClean, S.: Evaluation of system measures for incomplete relevance judgment in IR. In: Larsen, H.L., Pasi, G., Ortiz-Arroyo, D., Andreasen, T., Christiansen, H. (eds.) FQAS 2006. LNCS (LNAI), vol. 4027, pp. 245–256. Springer, Heidelberg (2006)
13. Zobel, J.: How reliable are the results of large-scale information retrieval experiments. In: Proceedings of ACM SIGIR Conference, Melbourne, Australia, pp. 307–314 (August 1998)

5 Appendix

A. List of 53 runs selected in the TREC 9 Web track

01. apl9all;	02. apl9lt;	03. apl9ltdn;	04. apl9t;
05. apl9td;	06. apl9tdn;	07. CWI0002;	08. Flab9atd2N;
09. Flab9atdN;	10. Flab9atdnN;	11. Flab9atN;	12. hum9td4;
13. hum9tde;	14. hum9tdn;	15. hum9te;	16. iit00t;
17. iit00td;	18. iit00tde;	19. iswtd;	20. iswtdn;
21. jscbt9wcl1;	22. jscbt9wcl2;	23. jscbt9wll1;	24. jscbt9wll2;
25. jscbt9wls1;	26. jscbt9wls2;	27. Mer9Wt1;	28. Mer9Wtnd;
29. NEnm;	30. NEnmLpas;	31. NEnmLsa;	32. NENRtm;
33. NENRtmLpas;	34. NENRtm;	35. pir0Watd;	36. pir0Wt1;
37. pir0Wtd2;	38. pir0Wttd;	39. pir0WTTD1;	40. ric9dpx;
41. ric9dpxL;	42. ric9dsx;	43. Sab9web2;	44. Sab9web3;
45. Sab9web4;	46. Sab9web5;	47. Scai9Web1;	48. Scai9Web2;
49. Scai9Web4;	50. tnout9f1;	51. tnout9t2lk50;	52. UCCS3;
53. UCCS4			

B. List of 34 runs selected in the TREC 2001 Web track

01. apl10wc;	02. apl10wd;	03. flabxt;	04. flabxtd;
05. flabxtdn;	06. flabxtl;	07. fub01be2;	08. fub01idf;
09. fub01ne;	10. fub01ne2;	11. hum01tdlx;	12. iit01tde;
13. jscbtawtl1;	14. jscbtawtl2;	15. jscbtawtl3;	16. jscbtawtl4;
17. kuadhoc2001;	18. Merxtd;	19. msrcn2;	20. msrcn3;
21. msrcn4;	22. ok10wtnd0;	23. ok10wtnd1;	24. pir1Wa;
25. pir1Wt1;	26. pir1Wt2;	27. posnir01ptd;	28. ricAP;
29. ricMM;	30. ricMS;	31. ricST;	32. uncvslm;
33. uncvsmm;	34. UniNEn7d		

Expanding Sensor Networks to Automate Knowledge Acquisition*

Kenneth Conroy¹, Gregory C. May¹, Mark Roantree², and Giles Warrington¹

¹ CLARITY: Centre for Sensor Web Technologies, Dublin City University

² Interoperable Systems Group, School of Computing, Dublin City University, Glasnevin, Dublin 9, Ireland

Abstract. The availability of accurate, low-cost sensors to scientists has resulted in widespread deployment in a variety of sporting and health environments. The sensor data output is often in a raw, proprietary or unstructured format. As a result, it is often difficult to query multiple sensors for complex properties or actions. In our research, we deploy a heterogeneous sensor network to detect the various biological and physiological properties in athletes during training activities. The goal for exercise physiologists is to quickly identify key intervals in exercise such as moments of stress or fatigue. This is not currently possible because of low level sensors and a lack of query language support. Thus, our motivation is to expand the sensor network with a contextual layer that enriches raw sensor data, so that it can be exploited by a high level query language. To achieve this, the domain expert specifies events in a traditional event-condition-action format to deliver the required contextual enrichment.

1 Introduction

Many new applications employ sensors or networks of sensors to automatically monitor and generate reports and analysis across domains. Increasingly, elite sports men and women are monitored to determine the effects of various training sessions on their bodies. Multiple heterogeneous sensors are often deployed to discover physiological or biological information generated during the activity. As these sensors generate output in unstandardised and proprietary formats, examining it to identify key events or properties involves time consuming examination of multiple files. Manual alignment, integration and the application of context from which this data was gathered is required to aid with querying the information.

These issues can be demonstrated by examining a sport such as cycling. Laboratory based cycling experiments attempt to quantify certain aspects of the effect of cycling on the participant. This is facilitated by gathering data such as *power output* (a measure of work created by the cyclist in order to overcome the forces against them, such as gradient, drag, etc.), *cadence* (a measure of the

* This work is supported by Science Foundation Ireland under grant 07/CE/I1147.

number of times a pedal revolution is performed per minute) and heart rate, among other factors. By measuring these in a laboratory based environment, it is possible to generate a dataset that can be specific to the question being asked by researchers and free of external artefacts. Many different systems exist to test cyclists under laboratory conditions while attempting to recreate the specific demands of cycling, with each *cycling ergometer* (a machine designed to replicate cycling in a measurable and repeatable manner) generating and measuring its resistive force in a different manner. However, this can lead to significant differences between ergometers. For the purpose of reliability in testing, an athlete must repeat tests on the same ergometer, under the same environmental conditions, and in the same training state. This will not eliminate all the changes from test to test, but will reduce the error from testing on dissimilar systems.

Scientists tend to prefer field-testing rather than laboratory testing, as it provides additional environmental factors which can effect performance. However, field tests for absolute physiological values tend to be less exact than laboratory based tests and are logistically more difficult to perform. Depending on the activity, there are many different factors that can predict eventual performance during the event. These predictors can be physiological, environmental, or equipment specific. Measurement of physiological factors is generally done via heart rate monitoring, power output measurement, respiration, and psychological scales. The information gained on physiological performance factors can give insight into how an athlete is performing during the training session, race, or event in which they are partaking. Over repeat measures it can be possible to track changes in performance and fitness of the athlete. By sensing physiological, environmental, and equipment changes and how they affect each other we are able to get a greater understanding of the changes that are occurring in both racing and training. This can potentially allow the development of targeted training sessions to investigate aspects of race performance.

1.1 Motivation

Over the past decade cycling has undergone a surge in technology aimed at the measurement and analysis of training and racing. Due to its repetitive and prolonged nature, it is possible to measure many factors during cycling once a sensor is available to monitor the variable required. Technological advances have allowed sensors and computers to reduce in size and weight dramatically bringing previously laboratory based tools to the general market. Technologies such as *power measuring* and GPS systems are now light enough for competitive cyclists to apply them on their bicycles. Although some of these systems integrate several sensors with one unit, many do not. This generates a problem when several different sensor sets are needed to determine the information needs described above. As cyclists are ever concerned with gaining a competitive edge, a system that will allow them to combine and investigate the data gathered from several sources is crucial to cyclists, their coach, and the scientists who can interpret the data. Thus, the goal is to provide a means of facilitating high level queries across all of these low level devices.

1.2 Contribution

In simple terms, data can be queried if we develop a protocol to transfer it into a relational database or encode the data in XML. However, through working with both exercise physiologists and cyclists, we discovered that their information needs could not be met (queries could not be expressed) with a process of supplying structure and low level semantics to sensor data. Instead, a more complex layer of contextual enrichment was required to prepare sensor data for high level query languages. Furthermore, this contextual enrichment must be specified by end users and not by computer scientists. In this paper, we present a framework and methodology for automated processing of sensor data so that it can be queried using a standard query language. While this method uses XML to provide the structure for sensor data, it is the end user (domain expert) who can add semantics to the data through the specification of data mining rules. By working closely with end user scientists, we evaluate our system by meeting the information needs of the end user, allowing them to specify how data repositories are enriched with context data, and by reducing the query execution times as a result of the contextual enrichment process.

1.3 Structure

The structure of the paper is as follows: §2 introduces cycling, the domain in which our system was deployed and provides an overview of the EventSense system architecture, with the Context Profiles explored in detail in §3. §4 details our experimental evaluation and results, and in §5 we present related research. §6 details our conclusion and our current work.

2 User Requirements and Operating Architecture

In this section, we present the user requirements in the form of a query set, defined by the end users. Queries 1 to 5 in Table 1 can be expressed using XQuery but the remaining queries are more complex, difficult to express and may require long calculation times. We will then describe the architecture used enable the exercise physiologists to extract the required information.

In general, the system must collect data from several independent sources, synchronise the data, and structure the data in some manner. It must also provide a facility for defining and applying event rules specific to a particular domain. In Table 1, this includes the hill classification, and complex accelerometer based algorithms for pedal cadence/vector/force and braking activity. The system needs to work in a context driven environment where the user can specify if the data comes from a training session, race, or the laboratory.

2.1 The EventSense Architecture

Figure 1 illustrates the architecture of our proposed solution to sensor data management. The remainder of this section details the individual processors

Table 1. Sample Query Set

Queries	
1.	Find total amount of time spent above 250W (Power-measuring)
2.	Find Heart Rate for each occurrence above 250W (Power-measuring)
3.	Find total amount of time spent above 165BPM (Heart rate-measuring)
4.	Calculate average heart rate spent above 200W
5.	Find the total amount of time where pedal pivot = 'pivot_range_1'
6.	Find 'best intervals' for highest '1minute' heart rate and return values for distance covered
7.	Find the average performance factors (Power/Heart Rate/Speed) for each gradient of type='hill'
8.	Find the average Power value when pedal vector magnitude = 'peak'
9.	Find the average speed when braking activity = 'none'
10.	Find all occurrences where gradient_profile = 'flat' and cycle cadence = 'cadence_range_1'

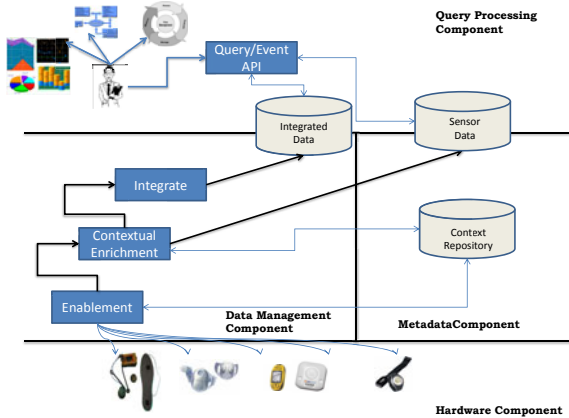


Fig. 1. EventSense System Architecture

involved to close the gap between the requirements of the domain expert and the initial format of the sensors output. These processors are discussed briefly and in the following section, we discuss Contextual Enrichment in greater detail - as this is the main focus of the paper.

Sensor Enablement. Sensors output data in a simple plaintext format. The common approach to analysing data requires considerable manual effort using spreadsheet tools to compare results across multiple sensor files. The common approach to automation is to build proprietary wrappers for every sensor to store data in a database format. This requires new wrappers for new sensors and database reengineering each time the structure of the output differs. Sensor Enablement is a form of structural enrichment whereby we convert all sensor data to a standard XML format using simple user defined templates. One can define how sensor output should be interpreted and structured using sensor profiles,

and automatically apply data transformations based on this information. The role of the Sensor Enablement processor is to generate machine queryable XML files.

Contextual Enrichment. Following Sensor Enablement, basic queries can be performed using XQuery. However, many of the queries listed in Table 1 cannot be expressed at this point as the data lacks the necessary semantics. To reduce this semantic gap, the system uses Participant and Activity Profiles to understand the deployment domain (or activity), the sensors, participants, and key events. Essentially, these are metadata constructs used to describe any object or person measured by the sensor. In the following section, we describe how the specification of event detection definitions can mine for the information necessary to end user queries.

Integration. The querying needs of the coaches and sports scientists are extensive, ranging from basic sensor analysis and comparison to anthropometric based analysis of participants with multiple sensors deployed while engaged in some activity. Some information such as sensor data is extensive and specific to a certain time span, or geographic location whereas much physiological or biological data rarely changes. In general, a single sensor cannot meet information needs and multiple sources of evidence must be integrated to provide both results and high levels of accuracy. For the current experiments used in this paper, sensors were manually synchronised and this processor was not used. However, analysis of sporting events is often chaotic and synchronisation of inexpensive sensors devices cannot be guaranteed. Thus, integration is an important part of current work.

Query Interface. As all the sensor data is converted to XML during sensor enrichment, queries can be expressed in XQuery or XPath. As neither of these languages are intuitive to non-computing users, a view based system is currently employed [5] which also offers optimisation features for high volume datasets. However, Contextual Enrichment is an important enabler for query processing as will be shown in the next section.

3 Context Profiles and Event Mining

Context Profiles provide genericity to the system and thus, facilitate heterogeneity. Individuals will have different physiological characteristics, activities will have different timings, layouts and formats, and sensors will come and go, bringing new information and heterogenous structures. The activity in which participants are being measured provides the widest range of heterogeneities. For this reason, it receives a more detailed discussion here, including how it can be used to extract new knowledge from the sensor database, that can later be exploited by the query processor. All profiles and function descriptions (discussed later) are stored in the System Repository.

3.1 Sensor, Activity and Participant Profiles

These three profiles are similar in nature. The sensor Profile allows for different sensors to be introduced at any point, providing their output is described using a template. The Activity Profile defines the activity or domain in which a set of sensors were deployed. Some of this information is standard for each activity, such as the start time, the sport involved and the list of sensors deployed. In addition, the Activity Profile defines the key elements of a deployment that are relevant for that sport and particular deployment. The Participant Profile provides the anthropometric data valid for a user at the time of deployment. There can be many participants, each with their own profile, in an activity.

3.2 Event Definition

The key component of contextual enrichment are the Event Definitions. These enable the end user to highlight important events during exercise activity given the domain algorithms required for a certain set of sensor data. These algorithms are defined by the end users. The Event Definition uses the traditional event-condition-action format with a sample event is shown in Example 1. The key elements are the **Event**, which has **Condition** and **Action** sub-elements. There may be multiple **Condition** elements joined by logical operators and any number of update **Action** elements.

Example 1. Terrain Classification

```

<Cycling_Events>
  <Event_Terrain_Classification_steep_climb>
    <Condition>
      <GarminGPS>
        <long ge 53.12714779087178>
        <long le 53.13754992640523>
        <lat le -6.29089645593262>
        <lat ge -6.31175331323243>
      </GarminGPS>
      <Logical_Operator= "OR" />
      <GarminGPS>
        <long ge 53.12714779087178>
        <long le 53.12714879587177>
        <lat le -6.28553256846468>
        <lat ge -6.31175331323243>
      </GarminGPS>
    </Condition>
    <Action>
      UPDATE <GarminGPS><Terrain> WITH <steep_climb>
    </Action>
  </Event_Terrain_Classification_steep_climb>
  <Event_Terrain_Classification_long_climb>
    <Condition>
      <GarminGPS>
        <long ge 53.09035966189816>
        <long le 53.13765290525642>
        <lat le -6.22017196862793>
        <lat ge -6.31192497460938>
      </GarminGPS>
    </Condition>
    <Action>
      UPDATE <GarminGPS><Terrain> WITH <long_climb>
    </Action>

```



```

</Event_Terrain_Classification_long_climb>
<Event_Terrain_Classification_long_descent>
  <Condition>
    <GarminGPS>
      <long ge 53.16354424912001>
      <long le 53.18638566546003>
      <lat le -6.29347137658692>
      <lat ge -6.29510215966797>
    </GarminGPS>
  </Condition>
  <Action>
    UPDATE <GarminGPS><Terrain> WITH <long_descent>
  </Action>
</Event_Terrain_Classification_long_descent>
...
</Cycling_Events>

```

In Example 1, the terrain corresponding to the GPS ranges are known to be *steep climb* sections of a race or training session. A logical OR operator ties two ranges of GPS values satisfying the steep climb criteria. There are two ranges because the race or training session has taken place where the steep climb is not always in the same direction geographically, and the GPS values do not uniformly increase or decrease. It is standard practise for cycling based domain experts to split the climb into two or more segments to allow this definition. If the GPS sensor values for latitude and longitude match the criteria in the condition, a <steep_climb> element is encoded within the <terrain> element of the sensor data file, as specified by the action.

Example 2. Vector Magnitude Classification

```

<Cycling_Events>
  <Event_VectorMagnitude_Classification_low>
    <Condition>
      <FnVectorMagnitude>
        <result le 500>
      </FnVectorMagnitude>
    </Condition>
    <Action>
      UPDATE <GT3XAccelerometer><VectorMagnitude> WITH <low>
      UPDATE <GT3XAccelerometer><VectorMagnitude> WITH <value>&result</value>
    </Action>
  </Event_VectorMagnitude_Classification_low>
  <Event_VectorMagnitude_Classification_average>
    <Condition>
      <FnVectorMagnitude>
        <result gt 500>
        <result le 1500>
      </FnVectorMagnitude>
    </Condition>
    <Action>
      UPDATE <GT3XAccelerometer><VectorMagnitude> WITH <average>
      UPDATE <GT3XAccelerometer><VectorMagnitude> WITH <value>&result</value>
    </Action>
  </Event_VectorMagnitude_Classification_average>
  <Event_VectorMagnitude_Classification_high>
    <Condition>
      <FnVectorMagnitude>
        <result gt 1500>
      </FnVectorMagnitude>
    </Condition>
    <Action>
      UPDATE <GT3XAccelerometer><VectorMagnitude> WITH <high>
      UPDATE <GT3XAccelerometer><VectorMagnitude> WITH <value>&result</value>
    </Action>
  </Event_VectorMagnitude_Classification_high>

```

```

    </Action>
  </Event_VectorMagnitude_Classification_high>
</Cycling_Events>

```

It is also necessary to support the use of functions to explicitly define complex algorithms, the results of which can be used as part of the condition. Example 2 shows the result of a function *FnVectorMagnitude* (which detects the direction and force of the power produced by the cyclist, allowing for the detection of the part of the pedal stroke at a given point in time) being used as part of the condition. In effect, we treat the output from the sensor and function in an identical manner. We support operators EQ, LT (less-than), GT, GE (greater-than-or-equal-to) and GT. The action is always an update of a sensor data file.

A simple user interface to define events means that the user is only required to select from the list of sensors or functions; the relevant properties, decide on the criteria for satisfying an occurrence of an event, and define what to update. Functions allow advanced algorithms to be applied which could not have been applied using XQuery alone.

4 Experiments and Evaluation

Experiments were run on identical servers with a 2.66GHz Intel Core2 Duo CPU and 4GB of RAM. The aim of the experiments is to compare query times on the contextually enriched data with equivalent queries on data which is only structurally enriched. We also measure the time taken for the once-off contextual enrichment, and illustrate the comparative ease of querying for the encoded domain events.

Table 2. Sample Event Detection Execution times

Filename (Event)	Size	Values	Enabled	Enriched	Result Size
1 wickm.xml (Strong Cadence)	3MB	17,798	178ms	82ms	2,111
2 raim.xml (Strong Cadence)	30MB	65,536	399ms	150ms	7,631
3 wickm.xml (Low Vector Magnitude)	3MB	17,798	n/a	104ms	5,652
4 raim.xml (Low Vector Magnitude)	30MB	65,536	n/a	374ms	29,490
5 wickgps.xml (Steep Climb Terrain)	150kb	655	77ms	75ms	49

A summary of the experiments is presented in Table 2. Two accelerometer sensor data files were queried to detect all occurrences of a low vector magnitude, and all occurrences of a strong cadence. This was performed twice to detect the cadence, once on the enabled data, where the cadence requirement is included as part of an XQuery expression, and secondly the query is performed following contextual enrichment, using a simple XQuery expression to detect occurrences of a strong cadence.

Due to its complex nature, the algorithm for Vector Magnitude cannot be queried using XQuery and thus, the query for low vector magnitude was performed on the contextually enriched data only. We chose two files to query, one

hour-long file representing the accelerometer deployed in an hour long mountainous time trial (*wickm*), the second has values from an 18-hour long ultra endurance race. As shown in the table, the query time for detecting cadence is significantly reduced for the contextually enriched data. Of the 17,798 entries in *wickm.xml*, 2,111 matched the criteria of a strong cadence. In the larger *raim.xml* file, 7,631 of 65,536 entries correspond to a strong cadence. Following contextual enrichment, we can detect vector magnitude of type = Low. Due to the increased number of results matching the criteria, the query time is longer than the query for strong cadence.

The GPS based query is also performed both before and after contextual enrichment. The time taken to evaluate is similar in both cases due to the relatively small size of the input file, *wickgps.xml*. The main benefit of including GPS based ranges as event definitions is that it allows the end user to specify the important segments of the session which is applied directly to the data and made simple to query. GPS coordinates are bulky and having to pass them as part of a complex query to detect relevant segments of a session increases the potential for error. As GPS coordinates differ for every environment, it is necessary for the end user to have access to defining these boundaries efficiently.

Table 3. Sample Enrichment Times

Event	Filename	Sensor	Time
1 Cadence Classification	wickm.xml	GT3X Accelerometer	1,195ms
2 Cadence Classification	raim.xml	GT3X Accelerometer	12,245ms
3 Vector Magnitude Classification	wickm.xml	GT3X Accelerometer	1,074ms
4 Vector Magnitude Classification	raim.xml	GT3X Accelerometer	11,309ms
5 Terrain Classification	wickgps.xml	Garmin GPS	114ms

The time taken to contextually enrich the rules into the sensor files is displayed in Table 3, where times for vector magnitude classification and cadence classification are proportional to the input filesize. While times can require up to 12,245ms for the 30MB file, the process needs only to be performed once.

In summary, the experiments demonstrate that enablement and enrichment, with their XML and semantic overheads, can be queried using high level query languages without significant overhead. The main evaluation comes from our collaborators, the exercise physiologists, who provide the datasets, specify the queries, and can now extract information independently, using events and an XQuery interface.

5 Related Research

[6] describes the approach to building OntoSensor, a prototype sensor knowledge repository compatible with evolving Sensor Web infrastructure. OntoSensor includes definitions of concepts and properties adopted in part from SensorML, the Web Ontology Language (OWL) [11] and extensions to IEEE Suggested Upper

Merged Ontology (SUMO) [8]. Sensor ontologies are used to establish a terminology for sensors, their properties, capabilities and services. OntoSensor has a number of advantages, including self-descriptive metadata embedded in the descriptions, which can be used in various sensor discovery and reasoning applications. OntoSensor illustrates a semantic approach to sensor description and provides an extensive knowledge model. However, this approach lacks a distinctive data description model to facilitate interoperable data representation for sensor observation and measurement data. Additionally, it does not facilitate the specification or inclusion of context by the end user.

In [1], the authors describe a semantic model for heterogeneous sensor data representation. A sensor data ontology is created based on the Sensor web Enablement (SWE) [7] and SensorML data component models. Semantic relationships and operational constraints are deployed in a uniform structure to describe the sensor data. The ontology based model allows machines to process and interpret the emerging semantics to create intelligent sensor network applications. However, this work is in an early stage of development, with many of its aims and goals yet to be implemented, whereas we have a working prototype system which facilitates interaction with domain experts and full query interface.

In [12], the authors represent context with varying granularity with a tuple consisting of an RDF triple defining the relationship, a lifespan and a conditional confidence value. This project aims to reduce uncertainty in context integration. The method used to achieve this is combining multiple sources of information and using a Bayesian approach to calculate conditional confidence values. This is useful for the target ubiquitous computing environment but is not suitable for an ever-changing set of events to be detected using multiple sensors in multiple locations.

In the core target domain of analysing sensor data corresponding to cyclists, there are a number of tools available which allow a limited analysis for sensor data. The most successful commercial application for analysing power meter data in the cycling domain is TrainingPeaks WKO+ [9]. An open source application, Golden Cheetah [4] can also be used to analyse cycling sensor data. Querying in WKO+ is limited to identifying the minimum/maximum/average data value for each stream for a lap-by-lap or specific time period defined by the user. Apart from the wattage analysis, no additional variables such as speed or current position can be applied as a filter. Querying is not supported by Golden Cheetah. In addition, Neither of these applications can support user defined events or context.

6 Conclusions

Sensor technology is used in many application areas now as a means of automated data generation and collection. However, the low level nature of these devices and the often complex query requirements of end users and specialists, means that a considerable gap exists between the information generation and end user queries. In this research, our goal was to minimise or even close that gap by allowing users to specify events that would lead to contextual enrichment of the data sources. Our system begins with an automatic process of basic enrichment which we refer

to as sensor enablement. At the next point in the architecture, the end users can influence the *level* and *type* of context by specifying a series of events. Before we introduced this step many queries were difficult to express and in some cases, it was not possible to express the more complex queries. As is typical in data warehouse systems, this also leads to an improvement in query processing times as the knowledge acquisition step provides partially executed queries.

Our current efforts are focused on the Integration Processor as we are currently limited to situations where each sensor can be synchronised against a common clock. As we begin to introduce sensors from outside our direct control, we must be able to auto-synchronise based on a set of algorithms we are currently developing. However, the delivery of a high-level interface for sensor data analysis provides a significant step forward for exercise physiologists where previous efforts required a manual analysis of spreadsheet data.

References

1. Barnaghi, P.M., Meissner, S., Presser, M., Moessner, K.: Sense and sens'ability: Semantic data modelling for sensor networks. In: Proceedings of the ICT Mobile Summit 2009 (2009)
2. Botts, M., Percivall, G., Reed, C., Davidson, J.: OGC Sensor Web Enablement: Overview and High Level Architecture. In: Nittel, S., Labrinidis, A., Stefanidis, A. (eds.) GSN 2006. LNCS, vol. 4540, pp. 175–190. Springer, Heidelberg (2008)
3. Dey, A.K.: Understanding and Using Context. *Personal Ubiquitous Computing* 5(1), 4–7 (2001)
4. GoldenCheetah (2011), <http://goldencheetah.org/>
5. Liu, J., Roantree, M., Bellahsene, Z.: A SchemaGuide for Accelerating the View Adaptation Process. In: Parsons, J., Saeki, M., Shoal, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 160–173. Springer, Heidelberg (2010)
6. Russomanno, D.J., Kothari, C., Thomas, O.: Building a sensor ontology: A practical approach leveraging ISO and OGC models. In: The 2005 International Conference on Artificial Intelligence, pp. 637–643. CSREA Press (2005)
7. Sensor Web Enablement (SWE) (2011), <http://www.opengeospatial.org/projects/groups/sensorweb>
8. Suggested Upper Merged Ontology (2011), <http://suo.ieee.org/SUO/SUMO/index.html>
9. TrainingPeaks WKO (2011), <http://www.peaksware.com/trainingpeaks-wko.aspx>
10. Wang, X., Dong, J.S., Chin, C., Hettiarachchi, S., Zhang, D.: Semantic Space: An Infrastructure for Smart Spaces. *IEEE Pervasive Computing* 3(3), 32–39 (2004)
11. Web Ontology Language (2011), <http://www.w3.org/TR/owl-features/>
12. Ye, J., McKeever, S., Coyle, L., Neely, S., Dobson, S.: Resolving uncertainty in context integration and abstraction: context integration and abstraction. In: ICPS 2008: Proceedings of the 5th International Conference on Pervasive Services, pp. 131–140. ACM, New York (2008)

Utilising the MISM Model Independent Schema Management Platform for Query Evaluation*

Cornelia Hedeler and Norman W. Paton

School of Computer Science, The University of Manchester
Oxford Road, Manchester M13 9PL, UK
{chedeler,norm}@cs.manchester.ac.uk

Abstract. Model Management, and its associated operators, provides generic means for dealing with multiple schemas and the mappings between them, for example, in the context of multiple heterogeneous data sources that need to be integrated. One example of a Model Management framework is the ‘Model Independent Schema Management’ (MISM) platform. In the context of MISM, algorithms and implementations of various operators have been proposed that act on a source-model independent metamodel. However, although the results on MISM indicate how to import and manipulate data from heterogeneous source types, to date no approach has been proposed to utilise MISM for querying across the multiple data sources. This paper presents SMql, a query language over the source-model independent supermodel, presents an algebra into which the query is translated and presents an approach for rewriting SMql queries into source-model-specific queries posed over the corresponding relational or XSD models of the data source to be queried. Thus this paper helps to complete the collection of problems that need to be addressed to allow source model-independent model management using universal models in the context of MISM.

Keywords: Model Management, Query Rewriting, Query Language.

1 Introduction

The vision of model management [5,6] was proposed to address the recurring issues that arise when dealing with data sources, whether multiple heterogeneous data sources that need to be integrated or a single data source with a continually evolving schema. In the case of multiple heterogeneous data sources, these could also be represented using different data models, e.g., relational or XSD. Model Management aims to provide generic operators that make it easier to manipulate schemas, that may be associated with, e.g., relational, object-relational or XML data sources, and the relationships between the schemas. With a view to obtaining data model independence, an approach has been proposed to represent models expressed in various different data models, within the same universal

* The work reported in this paper was supported by a grant from the EPSRC.

Table 1. Model-generic and model-specific constructs

Metaconstructs	Relational	XSD
Abstract		Root Element
Aggregation	Table	
StructOfAttributes		ComplexElement
Lexical	Column	Simple Element
Foreign Key	Foreign Key	Foreign Key

model, referred to as a supermodel. Utilising the benefits of data model independence, it has been suggested recently that dataspace management systems could utilise model management systems, e.g., for the integration of heterogeneous schemas or for dealing with evolving schemas [9]. This, however, requires model management systems to provide support for query evaluation across the various heterogeneous data sources.

A prominent example of a source-model independent approach is the ‘Model Independent Schema Management’(MISM) [14] framework. In MISM, implementations of various of the model management operators have been proposed, e.g., an extended version of ModelGen [3], which in addition to translating schemas from a representation in one model into an equivalent representation in another model is also able to translate the corresponding data. This, however, required the whole database to be loaded, making it only suitable as an off-line approach. This was addressed for relational data sources in a later proposal [2] in which the data translation rules presented previously are translated into executable SQL statements. The framework was later extended further by proposing definitions and implementations for Diff and Merge [1] over the supermodel.

However, even though MISM now provides support for managing and integrating schemas represented in various models, it still does not provide support for querying across the various data sources associated with the integrated schemas, whereas other model management platforms have been extended to provide support for query answering (e.g., [17,13]). This paper addresses this gap by presenting an approach for processing queries in the MISM framework. To do this we define a query language (*SMql*) and an algebra over the MISM supermodel and present an approach to evaluating *SMql* queries over relational and XML sources.

The remainder of the paper is organised as follows. Section 2 introduces the relevant components of the supermodel of the MISM platform, and Section 3 introduces the query language over the supermodel and the algebra. Section 4 describes the approach for query rewriting. Related work is presented in Section 5 and Section 6 concludes the paper.

2 Background

This section introduces the two levels of schema descriptions of MISM as proposed by Atzeni *et al.* [14]. The two levels are the model-specific description, which contains all the constructs required to represent schemas in a particular model (see the two UML diagrams in the bottom half of Figure 1 for relational

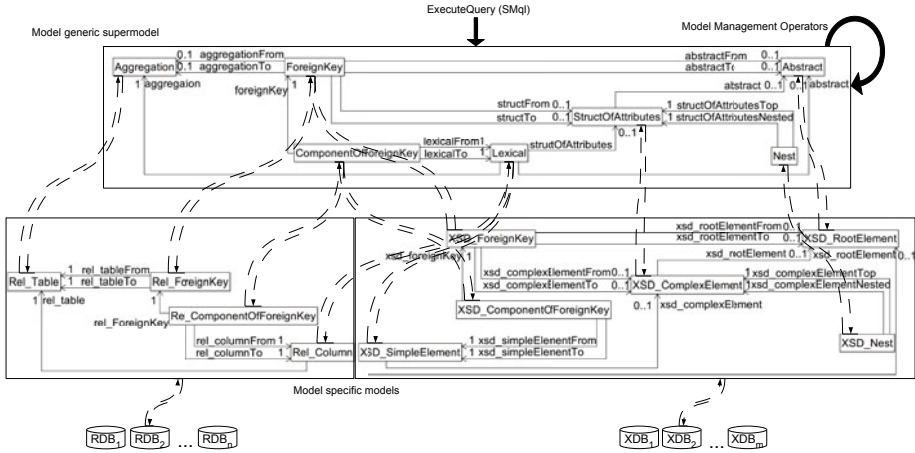


Fig. 1. Constructs in source-model independent supermodel and model specific models as well as the correspondences between them

and XSD, respectively) and the source-model independent supermodel, which uses a small set of model-generic constructs, so called *metaconstructs* [1] to represent model-specific constructs by aggregating over their similarities (see the UML diagram in the top half of Figure 1). Model management operators are defined over the constructs in the supermodel, which is depicted by the arrow in the top right corner of Figure 1. The UML diagrams in Figure 1 also include additional constructs that are required to represent all the information present in the models, e.g., *ComponentOfForeignKey* and *Nest*. By capturing both, the model-specific constructs and the model-generic representations of a schema, this approach is both model-independent and model-aware. Table 1 lists the model-generic metaconstructs and their corresponding model-specific constructs for relational and for XSD models. The dashed lines in Figure 1 from the model specific constructs to the model generic metaconstructs depict the corresponding constructs in the different models. The correspondences between the model specific and the model generic constructs are utilised during import of models, information that we later utilise in the opposite direction for query translation (depicted in Figure 1 by the dashed lines from the model generic constructs to the model specific constructs). As this paper focusses on relational and XSD models only, the remaining models that can be represented by the universal model have been omitted here, but are described in [4].

3 Query Language

This section introduces the query language *SMql*, a declarative query language inspired by SQL but defined over the constructs of the supermodel. A *SMql* query is of the form `SELECT l_1, \dots, l_n FROM c_1, \dots, c_m WHERE p` , where l_1, \dots, l_n is a project list of Lexicals, $c_1, \dots, c_m \in \{Abstract|Aggregation|StructOfAttributes\}$

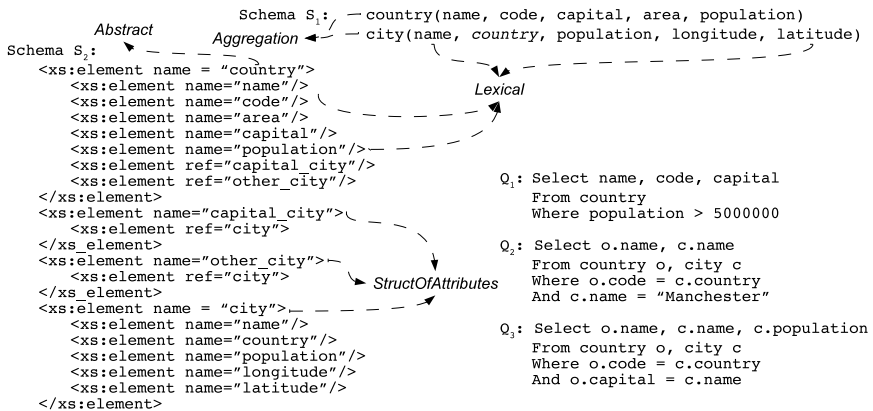


Fig. 2. Example schemas and queries in SMql

Table 2. SMql algebra

Operator	
SCAN(Abstract Aggregation StructOfAttributes)	→ Collection
REDUCE(Collection, {Lexical})	→ Collection
FILTER(Collection, Predicate)	→ Collection
JOIN(Left_Collection, Right_Collection, Predicate)	→ Collection
UNION(Left_Collection, Right_Collection)	→ Collection
EvaluateSQL(SQLQueryString, Predicate, {tuple})	→ {tuple}
EvaluateXQuery(XqueryString, Predicate, {tuple})	→ {tuple}

and p is a conjunctive predicate. Figure 2 shows two simplified schemas, S_1 of a relational data source and S_2 of an XSD data source and some example queries in *SMql*. The figure also shows the corresponding constructs in the supermodel for the two relational tables in S_1 , as well as the root element (*country*) and the complex elements (*capital_city*, *other_city*, *city*) in S_2 . The columns in S_1 and the simple elements in S_2 all correspond to *Lexical*, but for clarity not all those correspondences are shown in Figure 2.

SMql queries are translated into the algebra from Table 2 following standard translation schemes [10]. For example, query Q_1 in Figure 2 is translated into REDUCE(FILTER(SCAN(*country*), *population* > 5000000), {*name*, *code*, *capital*}) and query Q_2 is translated into REDUCE(JOIN(SCAN(*country* *o*), FILTER(SCAN(*city* *c*), *c.name* = ‘Manchester’), *o.code* = *c.country*), {*o.name*, *c.name*}). The UNION operator will be used later in the context of query unfolding whereas EvaluateSQL and EvaluateXQuery will be used later in the context of evaluation of the rewritten source-specific subqueries (see Section 4 for an example).

4 Query Rewriting

This section introduces the approach to rewriting a *SMql* query posed over constructs of the supermodel and expressed in the algebra shown in Table 2 into potentially multiple SQL or XQuery queries, respectively, depending on the constructs which are queried and their respective sources. For example, if a *SMql*

query is posed over a model that was generated using the model management operator Merge on two models, the query is expanded into a *SMql* query over constructs from potentially multiple source models using query unfolding [11] and associations between the source models and the merged model. The expanded *SMql* query is compiled into the algebra and optimised, and subqueries of the logically optimised plan that are associated with specific sources are then translated to the source specific query languages as described in Sections 4.1 and 4.2, whereby the left hand side input and the right hand side input of the UNION operator are treated as separate subqueries that are processed separately even if they are to be evaluated over the same source. The translated subqueries are passed to the operators EvaluateSQL and EvaluateXQuery, respectively, which can be parameterised with tuples and a Predicate, e.g., in the case of joins between different sources. Assume, for simplicity, that query Q_2 is posed over the merged schema of S_1 and S_2 (not shown) and that constructs with the same names in the three schemas are associated. The expanded algebra with the translated subqueries of Q_2 resulting from the use of these associations, query unfolding and the translation algorithms introduced in Sections 4.1 and 4.2 is shown in Figure 3.

Query rewriting is a two step process applied both for translating (portions of) a query into SQL and into XQuery. The first step is a recursive algorithm that traverses all operators in the query posed over the supermodel, and gathers all the information required for query rewriting in the corresponding data structures appropriate for the type of target query. In the second step, this information is utilised to generate a string representation of the target query. The process is described in more detail in the following.

The approach presented here only deals with syntax; dealing with semantics is beyond the scope of this paper. Once access has been provided to a source, the other model management operators provide techniques for manipulating the resulting integration model in ways that reflect semantic issues.

4.1 SMql Query over Supermodel into XQuery over XML

Based on the parts of an XQuery, namely the *Let*, *For*, *Where* and *Return* clauses, the following data structure is introduced to gather the information that is needed for each of the clauses.

An XQuery is a quadruple $\langle \text{let}, \text{for}, \text{where}, \text{return} \rangle$, where *let* is a map of variable names and references to source documents, *for* is a list of abstract/root element or structOfAttributes/complex elements, *where* is a list of conjunctive predicates and *return* is a list of fully qualified lexical/simple element names, either qualified with the name of the abstract|structOfAttributes / root|complex element the lexical belongs to or with the corresponding variable name. Both lists and maps support the operators *add* and *contains*. We assume here that the data source, or the source document of each instance i of a construct in the supermodel can be obtained by $i.\text{source}$.

We follow the two step process described briefly above, which consists of gathering the information for the various clauses of the XQuery by recursively

```

UNION(
  UNION(
    EvaluateXQuery(
      'for $o in $s/country
      for $c in $o/capital_city/city
      where $o/code = $c/country
      and $c/name = "Manchester"',
      null, null),
    EvaluateXQuery(
      'for $o in $s/country
      for $c in $o/other_city/city
      where $o/code = $c/country
      and $c/name = "Manchester"',
      null, null)),
  EvaluateSQL(
    'Select o.name, c.name
    from country o, city c
    where o.code = c.country
    and c.name = "Manchester"',
    null, null))

```

Fig. 3. Expanded and rewritten query Q_2

Require: s = query (fragment) to be translated, expressed in algebra introduced above

```

1: if s instance of SCAN(abstract c) | s instance of SCAN(structOfAttributes c) then
2:   if !s.let.contains(c.source) then
3:     s.let.add(c, c.source)
4:   end if
5:   s.for.add(c)
6: else if s instance of REDUCE(Collection c, {Lexical}) then
7:   for all Lexical l in {Lexical} do
8:     s.return.add(l)
9:   end for
10:   Translate2XQuery(c)
11: else if s instance of FILTER(Collection c, Predicate) then
12:   s.where.add(p)
13:   Translate2XQuery(c)
14: else if s instance of JOIN(Left_Collection lc, Right_Collection rc, Predicate) then
15:   s.where.add(p)
16:   Translate2XQuery(lc)
17:   Translate2XQuery(rc)
18: end if

```

Fig. 4. Translate2XQuery(s)

traversing the *SMql*-algebra (Algorithm shown in Figure 4) followed by the generation of a string representation of the XQuery utilising the gathered information (Algorithm shown in Figure 5). The algorithms presented are not the only way to organise the information and generate the corresponding XQuery, as there are several ways of expressing the same XQuery. For example, an equivalent XQuery to query Q_1 in Figure 2 posed over S_2 could be written as one of the two versions v1 or v2:

```

v1:for $c in doc("...")//country[population>"5000000"]
v2:let $s := doc("...") for $c in $s//country where $c/population > 5000000

```

We have decided to follow the structuring of the information and consequently of the XQuery that is somewhat related to the structure of a SQL query, i.e., **for** corresponds to **from**, **return** corresponds to **select**, **where** corresponds to **where**, and to capture only the reference to the source document in **let**, which results in queries structured as exemplified in v2.

To generate an XQuery that reflects the relationship between the structure of the supermodel and the structure of the document, we order all the (root and complex) elements that are gathered in `s.for` according to their hierarchical structure in the source document (line 5 of the algorithm shown in Figure 5). We differentiate between `rootElements` and `complexElements`, which correspond to different constructs in the supermodel (lines 7-12) when generating the path expression in the `for` clause. We assume in Figure 5 that the (variable) name of a construct c queried can be obtained through $c.name$.

Using the algorithms presented, the XQueries posed over S_2 that correspond to the example queries Q_1 , Q_2 and Q_3 are shown in Figure 6. The *let* and *return* clauses are omitted for XQueries Q_2 and Q_3 . As there are two alternatives for mapping `city` in the integration schema to `city` in S_2 , namely, `capital_city/city` or `other_city/city`, two subqueries are to be evaluated over the respective data source with schema S_2 and their results unioned. Both subqueries are shown in Figure 6.

4.2 SMql Query over Supermodel into SQL Query over Relational Model

Based on the three parts of an SQL query, namely the *Select*, *From* and *Where* clauses, the following data structure is introduced to gather the information that is needed for each of the three clauses.

A SQL query is a triple $\langle \text{select}, \text{from}, \text{where} \rangle$, where *select* is a list of fully qualified lexical/column names, either qualified with the name of the aggregation/table the lexical belongs to or with the corresponding variable name, *from* is a list of aggregations/tables, and *where* is a list of conjunctive predicates. Lists support the operator *add*. As the correspondences between model specific constructs and model generic constructs are trivial for relational data sources, the rewriting algorithm is straightforward.

In the first step, a recursive algorithm (omitted here due to space constraints; for the corresponding algorithm for XQuery see Algorithm 4) traverses the *SMql* algebra and places the appropriate information into the data structures corresponding to each part of the SQL query, e.g., predicates of `FILTER` and `JOIN` operators are added to `s.where`, and all lexicals in `REDUCE` are added to `s.select`.

In the second step of the query rewriting, the gathered information is translated into a string for evaluation. As this step is straightforward for SQL, the algorithm is omitted here.

5 Related Work

Various contributions on query rewriting for data integration have been made over the years (e.g., [11,14]). In addition, some of the model management platforms have been extended to support query evaluation (e.g., Automed [7] and GeRoMe [12]).

GeRoMe, utilises a role based metamodel in which multiple model-independent roles can be attached to each model-specific schema element thereby specifying

Require: s = query (fragment) to be translated with all information gathered in s .let, s .for, s .where, and s .return

```

1: String qs = new String("<result>")
2: for all (variableName v, document d) ∈ s.let do
3:   qs += "let $" + v + " := doc(" + d + ")"
4: end for
5: order all c's in s.for according to their hierarchical structure in the source document utilising
   the information captured in the supermodel
6: for all element c ∈ ordered s.for starting from top do
7:   if c instance of root element then
8:     qs += " for $" + c.name + " in $" + v + "/" + c.name
9:   else if s.c instance of complex element then
10:    qs += " for $" + c.name + " in $" + p.name + "/" + c.name
11:    where p = parent complex element of c
12:   end if
13: end for
14: if !s.where.isEmpty() then
15:   qs += " where "
16:   for all predicate p in s.where do
17:     if p of kind (simple element l1) (op) (simple element l2) then
18:       qs += "$" + c1.name + "/" + l1.name + op + "$" + c2.name + "/" + l2.name
19:       where c1 and c2 are the corresponding parent complex elements of simple elements l1
         and l2, respectively
20:     else if p of kind (simple element l) (op) (constant) then
21:       qs += "$" + c.name + "/" + l.name + op + constant
22:       where c is the corresponding parent complex element of simple element l
23:     end if
24:     if s.where.hasNext() then
25:       qs += " AND "
26:     end if
27:   end for
28: end if
29: qs += "return <tuple>"
30: for all simple element l in s.return do
31:   qs += "<" + c.name + "." + l.name + ">"
32:   qs += " fn:data($" + c.name + "/" + l.name + ")"
33:   qs += "</" + c.name + "." + l.name + ">"
34:   where c is the corresponding parent complex element of simple element l
35: end for
36: qs += "</tuple>"
37: qs += "</result>"
38: return queryString

```

Fig. 5. toXQueryString(s)

```

XQuery Q1:
<result>
let $s := doc("...")
for $o in $s/country
where $o/population > 5000000
return
  <tuple>
    <o.name>{fn:data($o/name)}</o.name>
    <o.code>{fn:data($o/code)}</o.code>
    <o.capital>{fn:data($o/capital)}</o.capital>
  </tuple>
</result>

```

```

XQuery Q2:
for $o in $s/country
for $c in $o/capital_city/city
where $o/code = $c/country
and $c/name = "Manchester"

for $o in $s/country
for $c in $o/other_city/city
where $o/code = $c/country
and $c/name = "Manchester"

```

```

XQuery Q3:
for $o in $s/country
for $c in $o/capital_city/city
where $o/code = $c/country
and $c/capital = $c/name

for $o in $s/country
for $c in $o/other_city/city
where $o/code = $c/country
and $c/capital = $c/name

```

Fig. 6. (Partial) XQueries corresponding to Q_1 , Q_2 and Q_3

its properties in detail [12]. At the language level, GeRoMe uses source-to-target extensional mappings which are based on second-order tuple generating dependencies (SO tgs) [8] that are specified over the schema elements and their roles [13] to express the relationships between heterogeneous schemas represented using different data models. A conjunctive query posed over an integration schema expressed using the same formalism as the extensional mappings, is rewritten into a query over the sources using composition of the conjunction of source-to-target mappings between the source schemas and the integration schema and the query itself. The predicates of the resulting query, which is expressed over all the source schemas, are partitioned according to the corresponding sources and are then translated into the corresponding source-specific query language (SQL or XQuery) to be evaluated [13]. In contrast, rather than using composition we have illustrated an approach for expansion of a query posed over an integration schema using query unfolding [11] and presented in detail the rewriting of *SMql* (sub-) queries that are associated with specific sources into the source-specific query languages (SQL and XQuery).

In contrast, Automed utilises a lower-level hypergraph data model consisting of edges, nodes and constraints to represent schemas expressed in heterogeneous data models including XML [15]. Relationships between different schemas are expressed by a number of low level transformations between them, e.g., removing a node or an edge, that can be combined to form more complex transformations. The approach is called both as view (BAV) and the transformations are specified in such a way that they are reversible and that both local as view (LAV) and global as view (GAV) mappings can be derived between an integration schema and source schemas from the BAV transformations [16,17]. A query over an integration schema or any of the source schemas can be expressed in Automed's IQL query language, a comprehension-based functional query language, that is reformulated into a query over the (other) sources schemas using a combination of LAV and GAV query processing techniques over the BAV transformations [17]. However, no detail is provided on how the IQL query posed over the source schemas is rewritten into the source-specific query languages, which is the main contribution of our approach presented here.

6 Conclusions

Complementing the model management platform MISM we have presented *SMql*, a query language and its algebra over the MISM supermodel. We have illustrated an approach for expanding queries over multiple sources and presented an approach for rewriting *SMql* queries into the corresponding source specific queries posed over the sources to be queried. To add query rewriting capabilities for other data models for which MISM already provides support, such as, the object-relational model, the same approach as presented here for XSD and the relational model can be followed, i.e., gather the information according to the structure of the corresponding query language and use the information on the correspondences between the model-specific model and the source-model independent supermodel to create the specific target query. To include other models

that are not yet supported by MISM, e.g. RDF, the MISM model will have to be extended first and then the approach described here followed.

References

1. Atzeni, P., Bellomarini, L., Bugiotti, F., Gianforme, G.: Mism: A platform for model-independent solutions to model management problems. *J. Data Semantics* 14, 133–161 (2009)
2. Atzeni, P., Bellomarini, L., Bugiotti, F., Gianforme, G.: A runtime approach to model-independent schema and data translation. In: *EDBT*, pp. 275–286 (2009)
3. Atzeni, P., Cappellari, P., Bernstein, P.A.: Model-independent schema and data translation. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Böhm, K., Kemper, A., Grust, T., Böhm, C. (eds.) *EDBT 2006*. LNCS, vol. 3896, pp. 368–385. Springer, Heidelberg (2006)
4. Atzeni, P., Gianforme, G., Cappellari, P.: A universal metamodel and its dictionary. *T. Large-Scale Data- and Knowledge-Centered Systems* 1, 38–62 (2009)
5. Bernstein, P.A., Halevy, A.Y., Pottinger, R.A.: A vision for management of complex models. *SIGMOD Record* 29(4), 55–63 (2000)
6. Bernstein, P.A., Melnik, S.: Model management 2.0: manipulating richer mappings. In: *SIGMOD Conference*, pp. 1–12 (2007)
7. Boyd, M., Kittivoravitkul, S., Lazanitis, C., McBrien, P., Rizopoulos, N.: Automed: A bav data integration system for heterogeneous data sources. In: Persson, A., Stirna, J. (eds.) *CAiSE 2004*. LNCS, vol. 3084, pp. 82–97. Springer, Heidelberg (2004)
8. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C.: Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.* 30(4), 994–1055 (2005)
9. Franklin, M.J., Halevy, A.Y., Maier, D.: From databases to dataspace: a new abstraction for information management. *SIGMOD Record* 34(4), 27–33 (2005)
10. Garcia-Molina, H., Ullman, J.D., Widom, J.: *Database Systems The Complete Book*, 2nd edn. Pearson International Edition, London (2009)
11. Halevy, A.Y.: Answering queries using views: A survey. *The VLDB Journal* 10(4), 270–294 (2001)
12. Kensché, D., Quix, C., Chatti, M.A., Jarke, M.: Gerome: A generic role based metamodel for model management. *Journal on Data Semantics* 8, 82–117 (2007)
13. Kensché, D., Quix, C., Li, X., Li, Y., Jarke, M.: Generic schema mappings for composition and query answering. *Data & Knowledge Engineering (DKE)* 68(7), 599–621 (2009)
14. Lenzerini, M.: Data integration: A theoretical perspective. In: *PODS*, pp. 233–246 (2002)
15. McBrien, P., Poulouvasilis, A.: A semantic approach to integrating xml and structured data sources. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) *CAiSE 2001*. LNCS, vol. 2068, pp. 330–345. Springer, Heidelberg (2001)
16. McBrien, P., Poulouvasilis, A.: Data integration by bi-directional schema transformation rules. In: *ICDE*, pp. 227–238 (2003)
17. McBrien, P., Poulouvasilis, A.: P2p query reformulation over both-as-view data transformation rules. In: Moro, G., Bergamaschi, S., Joseph, S., Morin, J.-H., Ouksel, A.M. (eds.) *DBISP2P 2005 and DBISP2P 2006*. LNCS, vol. 4125, pp. 310–322. Springer, Heidelberg (2007)

Mining Sequential Patterns from Probabilistic Databases by Pattern-Growth

Muhammad Muzammal

Department of Computer Science, University of Leicester, UK
mm386@mcs.le.ac.uk

Abstract. We propose a *pattern-growth* approach for mining *sequential patterns* from *probabilistic databases*. Our considered model of uncertainty is about the situations where there is uncertainty in associating an *event* with a *source*; and consider the problem of enumerating all sequences whose *expected support* satisfies a user-defined threshold θ . In an earlier work [Muzammal and Raman, PAKDD'11], adapted representative candidate generate-and-test approaches, GSP (breadth-first sequence lattice traversal) and SPADE/SPAM (depth-first sequence lattice traversal) to the probabilistic case. The authors also noted the difficulties in generalizing PrefixSpan to the probabilistic case (PrefixSpan is a pattern-growth algorithm, considered to be the best performer for deterministic sequential pattern mining). We overcome these difficulties in this note and adapt PrefixSpan to work under probabilistic settings. We then report on an experimental evaluation of the candidate generate-and-test approaches against the pattern-growth approach.

Keywords: Mining Uncertain Data, Mining complex sequential data, Probabilistic Databases, Novel models and algorithms.

1 Introduction

Agrawal and Srikant [14,2] defined the problem of *Sequential Pattern Mining (SPM)*, which involves discovery of frequent sequences of events in data with a temporal component; SPM has become a classical and well-studied problem in data mining [7,13,3]. In classical SPM, the database to be mined consists of tuples $\langle eid, e, \sigma \rangle$, where e is an *event*, σ is a *source* and eid is an *event-id* which incorporates a *time-stamp*. A tuple may record a retail transaction (event) by a customer (source), or an observation of an object/person (event) by a sensor/camera (source). All of the components of the tuple are assumed to be *certain*, or completely determined.

However, it is recognized that data obtained from a wide range of data sources is inherently uncertain [1]. This paper is concerned with SPM in *probabilistic databases* [15], a popular framework for modelling uncertainty. Recently several data mining and ranking problems have been studied in this framework, including top- k [18,5], frequent itemset mining (FIM) [14] and sequential pattern mining [11,12]. In [11] two kinds of uncertainty in SPM were formalized:

source-level uncertainty (SLU) and *event-level uncertainty (ELU)*. In SLU, the “source” attribute of each tuple is uncertain: each tuple contains a probability distribution over possible sources (*attribute-level uncertainty* [15]). As noted in [11], this formulation applies to scenarios where it is known that some customer made a specific retail transaction, but the identity of the customer who made that transaction is uncertain. This could happen because of incomplete customer details, or because the customer database itself is probabilistic as a result of “deduplication” or cleaning [7]. In ELU, the source of the tuple is certain, but the events are uncertain. This applies to several scenarios involving sensors in fixed locations detecting events using noisy techniques. In this case, either the existence of an event is uncertain (for example tracking endangered animals using sensors [8] or aggregating unreliable observations into events [9]) or the event’s existence is essentially certain, but its content is uncertain (for example, reading a numberplate using automated numberplate recognition or sequences of search terms that may have ambiguous meanings [11]).

Our contributions. In [12], efficient algorithms were proposed for the SPM problem in SLU probabilistic databases, under the *expected support* measure. These algorithms were based on the *candidate generate-and-test* approach, which is known to be relatively inefficient for classical SPM. In [12], it was noted that it is not straightforward to adapt the *pattern-growth* approach [13], which is usually the best for classical SPM, to the probabilistic case. In this paper, we overcome the obstacles mentioned in [12], and propose a pattern-growth method for the SPM problem in probabilistic databases. In classical SPM, pattern-growth works by performing L_1 computation on a *projected* database. The key contributions of this work are to formulate the analogue of a projected database in probabilistic settings, and to identify the appropriate L_1 computation to perform on the projected database. Unlike the deterministic case, it appears that pattern-growth for probabilistic SPM appears to require additional memory. We have implemented the new pattern-growth algorithm and present an experimental evaluation of the pattern-growth algorithm against the ones presented in [12]; this evaluation shows that although pattern-growth is superior to candidate generation in the deterministic settings, the picture is not as clear as for the probabilistic case.

2 Problem Statement

Classical SPM [14,2]. Let $\mathcal{I} = \{i_1, i_2, \dots, i_q\}$ be a set of *items* and $\mathcal{S} = \{1, \dots, m\}$ be a set of *sources*. An *event* $e \subseteq \mathcal{I}$ is a collection of items. A *database* $D = \langle r_1, r_2, \dots, r_n \rangle$ is an ordered list of *records* such that each $r_i \in D$ is of the form (eid_i, e_i, σ_i) , where eid_i is a unique event-id, including a time-stamp (events are ordered by this time-stamp), e_i is an event and σ_i is a source.

A *sequence* $s = \langle s_1, s_2, \dots, s_a \rangle$ is an ordered list of events. The events s_i in the sequence are called its *elements*. The *length* of a sequence s is the total number of items in it, i.e. $\sum_{j=1}^a |s_j|$; for any integer k , a k -sequence is a sequence of length k . Let $s = \langle s_1, s_2, \dots, s_q \rangle$ and $t = \langle t_1, t_2, \dots, t_r \rangle$ be two sequences. We say that s is a *subsequence* of t , denoted $s \preceq t$, if there exist integers $1 \leq i_1 < i_2 < \dots < i_q \leq r$

such that $s_k \subseteq t_{i_j}$, for $k = 1, \dots, q$. The *source sequence* D_i corresponding to a source i is just the multiset $\{e | (eid, e, i) \in D\}$, ordered by *eid*. For any sequence s , define its *support* in D , denoted $Sup(s, D)$ is the number of sources i such that $s \preceq D_i$. The objective is to find all sequences s such that $Sup(s, D) \geq \theta m$ for some user-defined threshold $0 < \theta \leq 1$.

Probabilistic Databases. We define an SLU *probabilistic database* D^p to be an ordered list $\langle r_1, \dots, r_n \rangle$ of records of the form (eid, e, W) where *eid* is an event-id, e is an event and W is a probability distribution over \mathcal{S} ; the list is ordered by *eid*. The distribution W contains pairs of the form (σ, c) , where $\sigma \in \mathcal{S}$ and $0 < c \leq 1$ is the confidence that the event e is associated with source σ and $\sum_{(\sigma, c) \in W} c = 1$. An example can be found in Table 1(L). The *possible worlds* semantics of D^p is as follows. A *possible world* D^* of D^p is generated by taking each event e_i in turn, and assigning it to one of the possible sources $\sigma_i \in W_i$. Thus every record r_i in D^p takes the form $r'_i = (eid_i, e_i, \sigma_i)$, for some $\sigma_i \in \mathcal{S}$ in D^* . The complete set of possible worlds is obtained by enumerating all such possible combinations. We assume that the distributions W_i associated with each record r_i in D^p are stochastically independent; the probability of a possible world D^* is therefore $\Pr[D^*] = \prod_{i=1}^n \Pr_{W_i}[\sigma_i]$. For example, a possible world D^* for the database of Table 1 can be generated by assigning event e_1 to Z with probability 0.3, events e_2 and e_4 to X with probabilities 0.7 and 0.6 respectively, and event e_3 to Y with probability 0.3, and $\Pr[D^*] = 0.3 \times 0.7 \times 0.3 \times 0.6 = 0.0378$.

Table 1. An SLU event database (L) transformed to p-sequences (R). Note that the events like e_1 (marked with † on (R)) can only be associated with one of the sources X, Y and Z in any possible world

eid	event	W		p-sequence
e_1	(a, c, e)	$(X : 0.1)(Y : 0.6)(Z : 0.3)$	D^p_X	$(a, c, e : 0.1)^\dagger(b, c, d : 0.7)(a, d, e : 0.2)$ $(b, c, e : 0.6)$
e_2	(b, c, d)	$(X : 0.7)(Y : 0.3)$		
e_3	(a, d, e)	$(X : 0.2)(Y : 0.3)(Z : 0.5)$	D^p_Y	$(a, c, e : 0.6)^\dagger(b, c, d : 0.3)(a, d, e : 0.3)$
e_4	(b, c, e)	$(X : 0.6)(Z : 0.4)$	D^p_Z	$(a, c, e : 0.3)^\dagger(a, d, e : 0.5)(b, c, e : 0.4)$

As a possible world is a deterministic instance of a given probabilistic database, concepts like the support of a sequence in a possible world do apply. The *expected support* of a sequence s in D^p is computed as follows:

$$ES(s, D^p) = \sum_{D^* \in PW(D^p)} \Pr[D^*] * Sup(s, D^*), \quad (1)$$

The problem we consider is:

Given an SLU probabilistic database D^p , determine all sequences s such that $ES(s, D^p) \geq \theta m$, for some user-specified threshold θ , $0 < \theta \leq 1$.

Observe that it is not feasible to use Eq. 1 directly due to the exponential number of possible worlds. Consider for example, we want to compute the probability with which source X supports a sequence $s = (a)(b)$ in the sample database of Table 1. There can be two ways in which s can be supported by source X i.e. either e_1 does support (a) and at least one of the events e_2 or e_4 support (b) or alternatively, e_1 does not support (a) but e_3 does support (a) , and e_4 supports (b) . The probability that X supports s is calculated as 0.196. Clearly, computing the source support probability this way is an expensive operation. In [12], a dynamic programming (DP) based algorithm was proposed to compute the source support probability $\Pr[s \preceq D_i^p]$, and it was shown that the ES of a sequence could be computed as follows:

$$ES(s, D^p) = \sum_{i=1}^m \Pr[s \preceq D_i^p] \tag{2}$$

3 Pattern-Growth Approach

We now describe our Pattern-Growth-Approach (PGA) that uses in essence the already proposed sub-routines, Dynamic Programming (DP) and fast L_1 computation [12]; and integrates it with the pattern-growth mechanism [13]. We first review some concepts:

p-Sequences. A p -sequence is analogous to a source sequence in classical SPM, and is a sequence of the form $\langle (e_1, c_1) \dots (e_k, c_k) \rangle$, where e_j is an event and c_j is a confidence value. An SLU database D^p can be viewed as a collection of p -sequences D_1^p, \dots, D_m^p , where D_i^p is the p -sequence of source i , and contains a list of those events in D^p that have non-zero confidence of being associated with source i , ordered by *eid*, together with the associated confidence (see Table 1(R)). Further, given a sequence $s = \langle s_1, \dots, s_q \rangle$ and an item x , s can either be extended by adding x as a separate element in s , $s \cdot \{x\}$ (S-extension) or by appending x to the last element in s , $\langle s_1, \dots, s_q \cup \{x\} \rangle$ (I-extension). For example, for $s = (a)(b)$ and $x = c$, S- and I-extensions of s are $(a)(b)(c)$ and $(a)(b, c)$ respectively.

Dynamic Programming. Let i be a source, $D_i^p = \langle (e_1, c_1), \dots, (e_r, c_r) \rangle$, and $s = \langle s_1, \dots, s_q \rangle$ be any sequence. Now let $A_{i,s}$ be the $(q \times r)$ DP matrix used to compute $\Pr[s \preceq D_i^p]$, and let $B_{i,s}$ denote the last row of $A_{i,s}$, that is, $B_{i,s}[\ell] = A_{i,s}[q, \ell]$ for $\ell = 1, \dots, r$. For $1 \leq k \leq q$ and $1 \leq \ell \leq r$, $A[k, \ell]$ will contain $\Pr[\langle s_1, \dots, s_k \rangle \preceq \langle (e_1, c_1), \dots, (e_\ell, c_\ell) \rangle]$, so $A[q, r]$ is the value $\Pr[s \preceq D_i^p]$. We set $A[1, \ell] = 1$ for all ℓ , $1 \leq \ell \leq r$ and $A[k, 1] = 0$ for all $1 \leq k \leq q$, and compute the other values row-by-row. For $1 \leq k \leq q$ and $1 \leq \ell \leq r$, define:

$$c_{k\ell}^* = \begin{cases} c_\ell & \text{if } s_k \subseteq e_\ell \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where $c_{k\ell}^*$ is the probability that the element s_k is contained in the event e_ℓ in source i ; If $s_k \subseteq e_\ell$, $c_{k\ell}^*$ is equal to the probability that e_ℓ is associated with source i , and 0 otherwise. Next, use the following recurrence:

$$A[k, \ell] = (1 - c_{k\ell}^*) * A[k, \ell - 1] + c_{k\ell}^* * A[k - 1, \ell - 1]. \quad (4)$$

Lemma 1. *Given a p -sequence D_i^p and a sequence s , by applying Eq. 4 repeatedly, we correctly compute $\Pr[s \preceq D_i^p]$.*

Fast L_1 Computation. It was shown by [12] that it was possible to compute all frequent 1-sequences in a single pass over the database. The procedure for this is as follows: Initialize two arrays F and G , each of size $q = |\mathcal{I}|$, to zero and consider each source i in turn. If $D_i^p = \langle (e_1, c_1), \dots, (e_r, c_r) \rangle$, for $k = 1, \dots, r$ take the pair (e_k, c_k) and iterate through each $x \in e_k$, setting $F[x] := (F[x] * (1 - c_k)) + c_k$. Once finished with source i , if $F[x]$ is non-zero, update $G[x] := G[x] + F[x]$ and reset $F[x]$ to zero (for each source i use a list structure to keep track of all the non-zero entries in F). Finally, for any item $x \in \mathcal{I}$, $G[x] = ES(\langle x \rangle, D^p)$.

PrefixSpan. PrefixSpan is based on the idea of pattern-growth, and works as follows: First, all frequent 1-sequences are discovered. It is argued that any of the frequent 2-sequences must begin with a frequent 1-sequence and therefore, the complete set of sequential patterns can be partitioned into as many subsets as the number of frequent 1-sequences where each 1-sequence is taken as a prefix. A *projected database* is a *smaller* databased based on some prefix (sequence). For example, in the sample database of Table 1(R), a (d) -projected database is $\{ \langle (a, d, e : 0.2)(b, c, e : 0.6) \rangle, \langle (a, d, e : 0.3) \rangle, \langle (-, e : 0.5)(b, c, e : 0.4) \rangle \}$. The subset of sequential patterns is mined by constructing the set of projected databases based on frequent 1-sequences and mining each recursively. For example, if (e) is a frequent 1-sequence in the above (d) -projected database, a $(d)(e)$ -projected database looks like $\{ \langle (b, c, e : 0.6) \rangle, \langle \rangle, \langle \rangle \}$. This recursive mining process continues until no more sequential patterns could be found. For details see [13].

It was noted in [12] that it is not correct to simply perform the fast L_1 computation on a projected database. For example, if an (a) -projected database contained two p -sequences $(b : 0.5)(b : 0.5)(a : 0.5)$ and $(b : 0.5)(a : 0.5)(b : 0.5)$, then when considering whether $(a)(b)$ is frequent, it is not correct to compute the expected support of (b) in the projected database (for example, both p -sequences above would give the same contribution – 0.75 – to the support of (b) in the projected database, but clearly their support for $(a)(b)$ is different). In this work, we show how these sub-routines could be put together to find all frequent sequential patterns using PGA.

3.1 Pattern-Growth Step

Pre-conditions

1. s is a previously discovered frequent sequence.
2. The list of sources i , where $\Pr[s \preceq D_i^p] > 0$ is available.
3. The $B_{i,s}$ arrays for all such sources i are also available.

Table 2. An example of computing the ES of all S-extensions of $s = (a)$ for source X in the sample database of Table 1. The gray row is the $B_{X,s}$ array. In the bottom half, the cells that changed in F' after processing the corresponding event are marked as gray. After processing source X , values in the final column are updated to G' .

D_X^p	$(a, c, e : 0.1)$	$(b, c, d : 0.7)$	$(a, d, e : 0.2)$	$(b, c, e : 0.6)$
(a)	0.1	0.1	0.28	0.28
(a)	0.0	0.0	0.02	0.020
(b)	0.0	0.07	0.07	0.196
(c)	0.0	0.07	0.07	0.196
(d)	0.0	0.07	0.076	0.076
(e)	0.0	0.0	0.02	0.176
	(i)	(ii)	(iii)	(iv)

Objective: To compute the ES of all the S- or I-extensions of s in one pass over the database, and thus discover all frequent extensions of s .

Steps: We consider the two cases of finding the S- and I-extensions of s in turn. We compute the frequent S-extensions of s as follows: Let i be a source, $D_i^p = \langle (e_1, c_1), \dots, (e_r, c_r) \rangle$, and $s = \langle s_1, \dots, s_q \rangle$ be any sequence. Initialize two arrays F' and G' , each of size $q = |I|$ to zero and consider each source i in turn. Then scan $B_{i,s}$ up-to the first non-zero entry e_k , and for every item x in e_ℓ , $k < \ell \leq r$, update $F'[x]$ as follows:

$$F'[x] := ((1 - c_\ell) * F'[x]) + (c_\ell * B_{i,s}[\ell - 1]) \quad (5)$$

We keep track of all the non-zero entries in $F'[x]$, and once finished with source i , update $G'[x] := G'[x] + F'[x]$ and reset $F'[x]$ to zero. After all the sources i are processed, all the entries in $G'[x] \geq \theta m$ are frequent S-extensions of s . An example of this computation is shown in Table 2.

For the I-extensions case, the initializations are the same as for the S-extensions case. For a sequence $s = \langle s_1, \dots, s_q \rangle$ and source i , when scanning $B_{i,s}$ up-to the first non-zero entry e_ℓ which means $s_q \subseteq e_\ell$, for every item x in e_ℓ that is not in s_q and is lexicographically greater than all items in s_q , update $F'[x]$ as follows:

$$F'[x] := (1 - c_\ell) * F'[x] + (B_{i,s}[\ell] - B_{i,s}[\ell - 1] * (1 - c_\ell)), \quad (6)$$

and apply Eq. 6 to all the events e_ℓ , $\ell \leq r$, where $s_q \subseteq e_\ell$ (or alternatively where the $B_{i,s}$ values change). After all the sources i are processed, all the entries in $G'[x] \geq \theta m$ are frequent I-extensions of s .

Pattern-Growth Algorithm. An overview of our pattern-growth algorithm is in Fig. 1. We first compute the set of frequent 1-sequences, L_1 (Line 3) (assume L_1 is in ascending order). For each 1-sequence x , first we compute the $B_{i,x}$

Algorithm 1. Pattern-Growth Approach

```

1: Input: SLU probabilistic database  $D^p$  and support threshold  $\theta$ .
2: Output: All sequences  $s$  with  $ES(s, D^p) \geq \theta m$ .
3:  $L_1 \leftarrow \text{ComputeFrequent-1-sequences}(D^p)$ 
4: for all sequences  $x \in L_1$  do
5:   Compute  $B_{i,x}$  arrays
6:   Call ProjectedDB( $x$ )
7: function ProjectedDB( $s$ )
8:  $L_S \leftarrow \text{Compute Frequent S-extensions}$  {fast  $L_1$  computation}
9:  $L_I \leftarrow \text{Compute Frequent I-extensions}$  {fast  $L_1$  computation}
10: Output all Frequent Sequences  $\{s$  extended with  $x$ , for all  $x$  in  $L_S$  and  $L_I\}$ 
11: for all  $x \in L_S$  do
12:    $t \leftarrow \langle s \cdot \{x\} \rangle$  {S-extension}
13:   Compute  $B_{i,t}$  arrays
14:   ProjectedDB( $t$ )
15: for all  $x \in L_I$  do
16:    $t \leftarrow \langle s_1, \dots, s_q \cup \{x\} \rangle$  {I-extension}
17:   Compute  $B_{i,t}$  arrays
18:   ProjectedDB( $t$ )
19: end function

```

arrays for each source (Line 5) and also keep track of all the sources where $\Pr[x \preceq D_i^p] > 0$ (projected database) and then, call the ProjectedDB(x) sub-routine (Line 6).

In the ProjectedDB sub-routine, we first compute all the frequent S- and I-extensions of s using the fast L_1 computation by applying Eq. 5 and Eq. 6 accordingly (Line 8 and 9). In step 10, output all the frequent S- and I-extensions of s computed in the previous steps. In steps 11-18, for every sequence t which is a frequent S- or I-extension of s , compute $B_{i,t}$ arrays and also keep track of all the sources where $\Pr[t \preceq D_i^p] > 0$, and call the ProjectedDB sub-routine recursively to mine all frequent sequential patterns.

Table 3. Number of DP computations (in millions) performed by each algorithm, for the set of experiments in Fig. 1 (L) and in Fig. 2 (R)

c10d10k	BFS+P	DFS+P	PGA
$\theta = 0.5\%$	3.141	3.199	1.494
$\theta = 1\%$	1.711	1.465	0.886
$\theta = 2\%$	0.879	0.781	0.487
$\theta = 4\%$	0.505	0.487	0.281
gazelle			
$\theta = 0.01\%$	0.777	0.375	0.261
$\theta = 0.02\%$	0.149	0.172	0.152
$\theta = 0.03\%$	0.073	0.129	0.122
$\theta = 0.04\%$	0.045	0.110	0.107

$C = 10, \theta = 1\%$	BFS+P	DFS+P	PGA
$D = 10K$	1.465	1.711	0.886
$D = 20K$	2.890	3.370	1.735
$D = 40K$	5.716	6.718	3.464
$D = 80K$	11.460	13.423	6.936
$D = 10K, \theta = 25\%$			
$C = 10$	0.100	0.111	0.081
$C = 20$	0.690	0.694	0.314
$C = 40$	13.044	13.891	4.868
$C = 80$	2353.677	2782.807	881.480

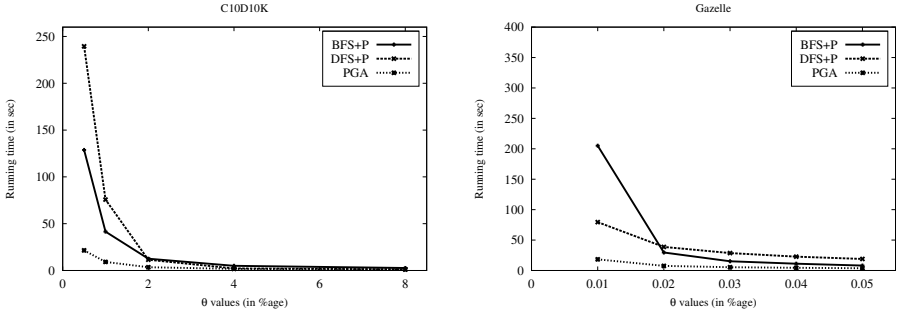


Fig. 1. Scalability of the three algorithms for decreasing values of θ , for synthetic dataset (C10D10K) (L) and for real dataset Gazelle (R)

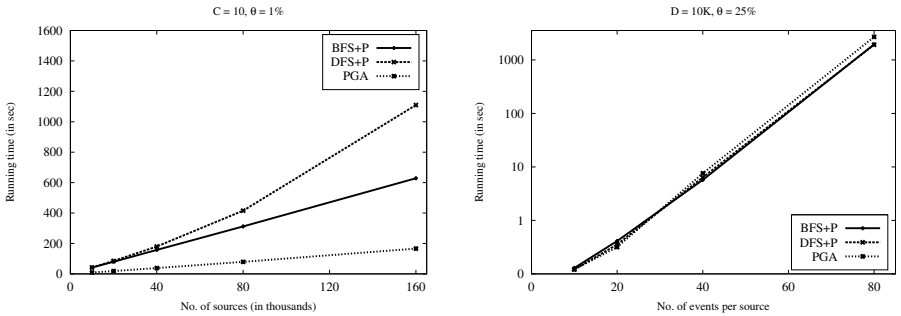


Fig. 2. Scalability for increasing number of sources D , with average number of events per source $C = 10$ and ES = 1% (L), and for increasing number of events per sources C with number of sources $D = 10K$ and ES = 25% (R)

4 Experimental Evaluation

In [12], the authors adapt GSP and SPADE/SPAM to yield a breadth-first (BFS) and a depth-first (DFS) algorithm, respectively. In addition, they also propose a probabilistic pruning technique to eliminate potential infrequent candidates without support computation, achieving an overall speedup. We therefore, choose two of the faster candidate generation variants from [12] i.e. BFS+P (breadth-first search with pruning) and DFS+P (depth-first search with pruning), and compare these with the Pattern-Growth Approach (PGA) proposed in this work.

Our implementations are in C# (Visual Studio .Net 2005), executed on a machine with a 3.2GHz Intel CPU and 3GB RAM running XP (SP3). We begin by describing the datasets used for experiments. Then, we demonstrate the scalability of the three algorithms. Our reported running times are averages from multiple runs. In our experiments, we use both real (*gazelle* from Blue Martini [10]) and synthetic (IBM Quest [2]) datasets. We transform these deterministic datasets to probabilistic form in a way similar to [14,18,12]; we assign probabilities to each

event in a source sequence using a uniform distribution over $(0, 1]$, thus obtaining a collection of p-sequences.

The real dataset Gazelle has 29369 sequences and 35722 events. For synthetic datasets, we follow the naming convention of [17]: a dataset named $CiDjK$ means that the average number of events per source is i and the number of sources is j (in thousands). Alphabet size is $2K$ and all other parameters are set to default. For example, the dataset $C10D20K$ has on average 10 events per source and $20K$ sources. We consider following three parameters in our experiments: number of sources D , average number of events per source C , and support threshold θ . We test our algorithms for increasing D , increasing C , and decreasing θ values by keeping the other two parameters fixed.

Scalability Testing. In the first set of experiments, we fix $C = 10$ and $\theta = 1\%$, and test the scalability of these algorithms for decreasing θ values. We report our results for synthetic dataset ($C10D10K$) in Fig. 1(L), and for real dataset (Gazelle) in Fig. 1(R). It can be seen that PGA performs better than both the candidate generate-and-test approaches for real as well as for synthetic dataset. The performance difference is more obvious for harder instances (at low θ values).

In another set of experiments, we test the scalability of these algorithms for increasing values of D by fixing $C = 10$ and $\theta = 1\%$ (Fig. 2(L)), and by fixing $D = 10K$ and $\theta = 25\%$, for increasing values of C (Fig. 2(R)). It can be seen that PGA performs better than the other two algorithms for increasing D (Fig. 2(L)). However, we do not see improvements for increasing C (Fig. 2(R)). We are currently investigating the reasons for this behaviour. Note that DFS+P processes only one S- or I-extension of s at a time, whereas in PGA all the extensions of s are processed simultaneously.

We also kept statistics about the number of DP computations for each algorithm (Table. 3). The datasets and support thresholds are the same as in Fig. 1 and Fig. 2. We observe that PGA performs the least number of DP computations consistently, as in PGA the $B_{i,s}$ arrays are computed only for the frequent sequences. As noted in [13] that candidate generation approaches suffer from an exponential number of candidates at low θ values, this cost is even higher in probabilistic case because of the $B_{i,s}$ arrays. Further, note that there is no need for keeping $B_{i,s}$ arrays for BFS in contrast with the PGA and therefore, PGA has additional memory needs.

5 Conclusions and Future Work

We have considered the problem of finding all frequent sequences by pattern-growth in SLU databases. We have evaluated PGA in contrast with the candidate generate-and-test approaches, and we observe that the PGA performs better than the candidate generation approaches in general. The speedup in running time can be seen for the real dataset, in particular at low θ values, and for the synthetic datasets as well when the source sequences are not very long or for low θ values. The statistics about the number of DP computations also show that the PGA performs the least number of DP computations consistently. We conclude

that PGA is generally efficient than the candidate generation algorithms. In future, we intend to investigate that why the performance difference is not so obvious when the source sequences are very long or for higher θ values, and whether PGA has similar behaviour for other real datasets.

References

1. Aggarwal, C.C. (ed.): *Managing and Mining Uncertain Data*. Springer, Heidelberg (2009)
2. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Yu, P.S., Chen, A.L.P. (eds.) *ICDE*, pp. 3–14. IEEE Computer Society, Los Alamitos (1995)
3. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: *KDD*, pp. 429–435. ACM, New York (2002)
4. Bernecker, T., Kriegel, H.P., Renz, M., Verhein, F., Züfle, A.: Probabilistic frequent itemset mining in uncertain databases. In: Elder, et al [6], pp. 119–128
5. Cormode, G., Li, F., Yi, K.: Semantics of ranking queries for probabilistic data and expected ranks. In: *ICDE*, pp. 305–316. IEEE, Los Alamitos (2009)
6. Elder, J.F., Fogelman-Soulié, F., Flach, P.A., Zaki, M.J. (eds.): *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Paris, France, June 28-July 1. ACM, New York (2009)
7. Hassanzadeh, O., Miller, R.J.: Creating probabilistic databases from duplicated data. *The VLDB Journal* 18(5), 1141–1166 (2009)
8. Hua, M., Pei, J., Zhang, W., Lin, X.: Ranking queries on uncertain data: a probabilistic threshold approach. In: Wang [16], pp. 673–686
9. Khoussainova, N., Balazinska, M., Suciú, D.: Probabilistic event extraction from RFID data. In: *ICDE*, pp. 1480–1482. IEEE, Los Alamitos (2008)
10. 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)
11. Muzammal, M., Raman, R.: On probabilistic models for uncertain sequential pattern mining. In: Cao, L., Feng, Y., Zhong, J. (eds.) *ADMA 2010, Part I*. LNCS, vol. 6440, pp. 60–72. Springer, Heidelberg (2010)
12. Muzammal, M., Raman, R.: Mining sequential patterns from probabilistic databases. In: Huang, J.Z., Cao, L., Srivastava, J. (eds.) *PAKDD 2011, Part II*. LNCS (LNAI), vol. 6635, pp. 210–221. Springer, Heidelberg (2011)
13. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: Mining sequential patterns by pattern-growth: The PrefixSpan approach. *IEEE Trans. Knowl. Data Eng.* 16(11), 1424–1440 (2004)
14. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.) *EDBT 1996*. LNCS, vol. 1057, pp. 3–17. Springer, Heidelberg (1996)
15. Suciú, D., Dalvi, N.N.: Foundations of probabilistic answers to queries. In: Özcan, F. (ed.) *SIGMOD Conference*, p. 963. ACM, New York (2005)
16. Wang, J.T.L. (ed.): *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008*, Vancouver, BC, Canada, June 10-12. ACM, New York (2008)
17. Zaki, M.J.: SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning* 42(1/2), 31–60 (2001)
18. Zhang, Q., Li, F., Yi, K.: Finding frequent items in probabilistic data. In: Wang [16], pp. 819–832

Smile: Enabling Easy and Fast Development of Domain-Specific Scheduling Protocols

Christian Tilgner¹, Boris Glavic², Michael Böhlen¹, and Carl-Christian Kanne³

¹ University of Zurich

² University of Toronto

³ University of Mannheim

{tilgner,boehlen}@ifi.uzh.ch, glavic@cs.toronto.edu,
kanne@informatik.uni-mannheim.de

Abstract. Modern server systems schedule large amounts of concurrent requests constrained by, e.g., correctness criteria and service-level agreements. Since standard database management systems provide only limited consistency levels, the state of the art is to develop schedulers imperatively which is time-consuming and error-prone. In this poster, we present *Smile* (declarative Scheduling MIddleware), a tool for developing domain-specific scheduling protocols declaratively. Smile decreases the effort to implement and adapt such protocols because it abstracts from low level scheduling details allowing developers to focus on the protocol implementation. We demonstrate the advantages of our approach by implementing a domain-specific use case protocol.

1 Introduction

Modern application servers handle large numbers of concurrent requests which have to be scheduled according to, e.g., correctness criteria like classical serializability or service-level agreements (SLAs). Standard database management systems (DBMSs) offer a limited amount of fixed consistency levels, do not provide sophisticated support for SLAs and, thus, often cannot be used to satisfy domain-specific scheduling requirements. The state of the art is to develop schedulers imperatively for applications like Amazon, Ebay or Yahoo [2,5] which yields fine-tuned schedulers satisfying the application's scheduling constraints. But procedural implementations of schedulers can be complex and difficult to understand, especially if the request types and correctness criteria are less well studied than, e.g., classic serializability. Adapting schedulers to evolving requirements results in expensive and error-prone re-implementations. With our approach we address these issues by leveraging a declarative language to implement schedulers which has been shown to be beneficial in previous work [13].

1.1 Banking Scenario

We use the following simplified banking scenario to illustrate the shortcomings of standard DBMSs with regard to non-standard scheduling requirements. A bank institute serves normal and premium customers holding bank accounts.

A domain expert defines the following constraints: (C1) Account data has to be accessed under strong consistency to obviate inconsistent states and (C2) Do not schedule requests for normal customers, if there are pending requests from premium customers. How can a scheduler developer implement these constraints? Constraint C1 can be realized with standard DBMSs by applying a high isolation level, but C2 is not supported by standard DBMSs. The alternative is to develop a new scheduler from scratch which is expensive and error-prone.

2 Smile: Declarative Scheduling Middleware

Smile, our declarative scheduling middleware prototype, allows the implementation of domain-specific scheduling constraints. Executable scheduling protocols are specified with few lines of code, paving the way for sophisticated and easy-to-reason-about scheduling protocols. Our approach is based on a generic formal framework called Oshiya¹ that models the scheduling state as a set of so-called *scheduling relations*, e.g., one relation stores the schedule produced so far. Scheduling logic is encapsulated in a set of declarative queries called *scheduling queries*. To produce a schedule (sequence of scheduling relation states), Smile schedules multiple requests at the same time by repeatedly executing the scheduling queries over the scheduling relations.

This approach has several advantages: (1) Smile abstracts from low level scheduling details that are independent of the scheduling constraints such as parallelism in the scheduler code, queueing of incoming requests, or managing (network) connections. Developers can focus on the protocol implementation (the scheduling queries) itself, which decreases the amount of code and the effort needed to implement or adapt schedulers. We developed scheduling queries for the strong two-phase locking (SS2PL) protocol [4] as well as for a data dependent, relaxed consistency protocol. (2) Smile’s underlying model allows to specify scheduling protocols close to their formal definition, facilitating reasoning over properties of protocol implementations such as verifying their correctness. For instance, we have proven the correctness of the scheduling queries implementing SS2PL [4]. (3) The separation of scheduling logic and scheduler implementation opens up interesting optimization opportunities that we plan to investigate in future work. E.g., using specialized execution engines to execute scheduling queries and controlling the trade-off between the time spent for scheduling requests and the time spent to execute them. (4) Scheduling sets of requests at the same time can improve the performance for large numbers of concurrent requests [3].

2.1 Oshiya Scheduling Model

In Oshiya [3], the scheduling state (requests to schedule and history information needed for scheduling decisions) is stored in three scheduling relations: *PendingRequests* (\mathcal{R}) buffers requests that have to be scheduled for execution. *RelevantHistory* (\mathcal{H}) stores prior executed requests in their execution order, modelling

¹ Oshiya refers to the passenger arrangement staff at Japanese train stations who help to fill a train by pushing people onto the train or guiding people to free railway cars.

the schedule generated so far. *Executable* (\mathcal{E}) buffers requests chosen for execution. Scheduling protocols are realized as three scheduling queries: $Q_{Schedule}$, $Q_{Revoked}$, $Q_{Irrelevant}$. Given previously executed requests, these queries determine in which order to execute pending requests. The scheduler state is advanced in iterative steps by applying a generic *scheduling algorithm* (shown in Fig. 2) that evaluates the scheduling queries over the current instances of the scheduling relations. The algorithm is the same for every protocol, but is parameterized by the protocol specific scheduling relations schemata and scheduling queries. Each *scheduler iteration* (while loop) performs the following steps: (1) Requests scheduled in the previous iteration are removed from \mathcal{R} . (2) Newly arrived client requests (\mathcal{N}) are added to \mathcal{R} . (3) $Q_{Revoked}$ determines transactions that have to be aborted since the requests cannot be executed due to constraint violations or blocking. (4) $Q_{Schedule}$ implements the scheduling protocol. It selects all requests from \mathcal{R} that can be executed in this iteration without violating the protocol constraints. (5) Requests in \mathcal{E} are executed and (6) added to \mathcal{H} . (7) $Q_{Irrelevant}$ identifies those requests from \mathcal{H} that are irrelevant for future scheduling decisions, and is used to prune \mathcal{H} so that it does not grow continuously.

2.2 Smile Architecture

The Smile prototype implements the Oshiya scheduling model outlined in the last subsection using three threads (*ClientWorker*, *Declarative Scheduler* and *Executor*), all running independently and continuously. The Smile architecture is shown in Figure 1 with arrows denoting data flow.

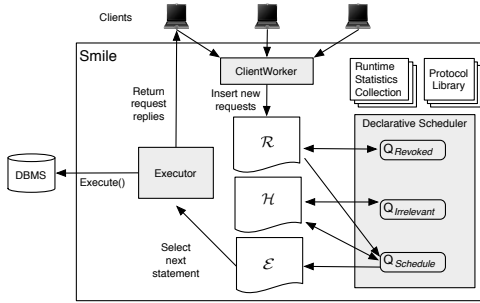


Fig. 1. Smile Architecture

```

 $\mathcal{H} = \mathcal{E} = \mathcal{R} = \emptyset$ 
while true do begin
1   $\mathcal{R} = \mathcal{R} - \mathcal{E}$ ;
2   $\mathcal{R} = \mathcal{R} \cup \mathcal{N}$ ;
3   $\mathcal{R} = \mathcal{R} - Q_{Revoked}(\mathcal{H}, \mathcal{R})$ ;
4   $\mathcal{E} = Q_{Schedule}(\mathcal{H}, \mathcal{R})$ ;
5  Execute( $\mathcal{E}$ );
6   $\mathcal{H} = \mathcal{H} \cup \mathcal{E}$ ;
7   $\mathcal{H} = \mathcal{H} - Q_{Irrelevant}(\mathcal{H})$ ;
end

```

Fig. 2. Smile Algorithm

ClientWorker. This thread manages client connections. The ClientWorker thread receives new requests from clients, buffers these client requests in a queue and periodically inserts them into \mathcal{R} as batch job (Step 2).

Declarative Scheduler. This thread performs request scheduling by periodically executing $Q_{Revoked}$, $Q_{Schedule}$ and $Q_{Irrelevant}$ (Steps 3, 4, 7).

Executor. The Executor thread is executing the scheduled requests located in \mathcal{E} against the DBMS by repeating the following steps: Retrieve the request with the smallest *ID* from \mathcal{E} , execute it against the back-end DBMS, return the request result to the client that has sent this request, and delete it from \mathcal{E} (Step 5).

Protocol Library. Smile offers a protocol library providing the scheduler developer with pre-cooked scheduling queries (e.g., for SS2PL). These scheduling queries can be used out of the box or as a starting point to develop domain-specific protocols. We expect developers to extend this library over time with their own protocol modules.

Runtime Statistics Collection. We let Smile gather statistics about the behaviour of its operations at runtime such as the cardinalities of \mathcal{R} , \mathcal{H} and \mathcal{E} and the execution times of the scheduling queries. In future work, we plan to expose this information to the scheduler developer for the use in the scheduling queries and let her provide policies for scheduling the execution of the Smile threads.

We developed strategies deciding when to pause a thread ensuring an efficient resource usage. E.g., running the Executor while \mathcal{E} is empty wastes resources.

2.3 Example: Use Case Implementation

We sketch the protocol implementation of the use case to illustrate the simplicity and conciseness of our approach. Scheduling queries are given as domain relational calculus (DRC) expressions. A simplified DRC formulation of $Q_{Schedule}$ implementing the use case constraints is:

$$Q_{Schedule} = \{S, C \mid is2PL(S, C) \wedge \exists C_2 (C_2 = 'premium' \wedge (is2PL(_, C_2) \Rightarrow C = 'premium'))\}$$

We use a declarative implementation of SS2PL to realize constraint C1. Predicate $is2PL(S, C)$ uses \mathcal{R} to determine all requests S with their customer class C that can be executed without violating the SS2PL constraints that have to hold for the generated schedule (requests in relation \mathcal{H}). We use S as a shorthand for the request related attributes of $is2PL$ (transaction ID etc.). Using Oshiya, we can implement scheduling constraint C2 as follows: If there exists at least one request of a premium customer ($C_2 = 'premium' \wedge is2PL(_, C_2)$), then only premium requests are selected by $Q_{Schedule}$ ($C = 'premium'$).

References

1. Alvaro, P., Condie, T., Conway, N., Elmeleegy, K., Hellerstein, J.M., Sears, R.: Boom Analytics: Exploring Data-Centric, Declarative Programming for the Cloud. In: EuroSys, pp. 223–236 (2010)
2. Cooper, B.F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.-A., Puz, N., Weaver, D., Yerneni, R.: PNUTS: Yahoo!'s Hosted Data Serving Platform. PVLDB 1(2), 1277–1288 (2008)
3. Tilgner, C.: Declarative Scheduling in Highly Scalable Systems. In: EDBT/ICDT Workshops, pp. 41:1–41:6 (2010)
4. Tilgner, C., Glavic, B., Boehlen, M., Kanne, C.-C.: Correctness Proof of the Declarative SS2PL Protocol Implementation. Technical Report IFI-2010.0008, University of Zurich, Department of Informatics (2010)
5. Vogels, W.: Data Access Patterns in The Amazon.com Technology Platform. In: VLDB, vol. 1 (2007)

On Integrating Data Services Using Data Mashups^{*}

Muhammad Intizar Ali¹, Reinhard Pichler¹,
Hong-Linh Truong², and Schahram Dustdar²

¹ Database and Artificial Intelligence Group, Vienna University of Technology
{intizar,pichler}@dbai.tuwien.ac.at

² Distributed Systems Group, Vienna University of Technology
{truong,dustdar}@infosys.tuwien.ac.at

Abstract. Mashups are applications that aggregate functionality, presentation, and/or contents from existing sources to create a new application. Contents are usually generated either using web feeds or an application programming interface (API). Both approaches have limitations as web feeds do not provide powerful data models for complex data structures and lack powerful features of database systems. On the other hand, API's are usually limited to a specific application thus requiring different implementations for each of the sources used in the mashups. We propose a query based aggregation of multiple heterogeneous data sources by combining powerful querying features of XQuery and SPARQL with an easy interface of a mashup tool for data sources in XML and RDF. Our mashup editor allows for automatic generation of mashups with an easy to use visual interface.

1 Introduction

The amount of structured and semi-structured data available on the internet has been steadily increasing and many companies are now providing their data publicly accessible through API's, querying interfaces, RESTful web services, or data services [1]. The rapid growth of Web 2.0 technologies has motivated many big companies to make their contents reusable for the creation of new applications using existing data. Many publicly accessible API's such as Google Maps¹, Amazon² and DBPedia³ are available for the users to generate their own new applications using their existing contents. A typical example of such a scenario is the combination of the list of hotels in a particular city with Google Maps to generate an interactive map of hotels or data collected from several news sites and merged together to provide a single access point to the user.

Mashups are web applications that consume the available data from third parties and combine/reuse them to build a new application. Mostly the contents are in the form of web feeds or API's. All the contents are combined either on client side using client-side scripts or on server-side using some available server-side technology such as ASP,

^{*} This work was supported by the Vienna Science and Technology Fund (WWTF), project ICT08-032.

¹ <http://maps.google.com>

² <http://www.amazon.com>

³ <http://www.dbpedia.org>

JSP, etc. Mashups are different from traditional web applications because they are usually dynamically created to serve a very specific and short lived task. Several mashup editors have been launched to encourage people to build new applications using the massive amount of publicly available contents. Yahoo Pipes⁴, Google Mashups⁵ and IBM Mashup Center⁶ are a few examples of the popular mashup editors. However, the limitation of existing mashup editors is that they focus only on web feeds or API's. These web feeds can represent simple information but lack the capability to represent or query data items provided by querying interfaces or data services [2]. On the other hand, API's are usually limited to a specific application thus requiring different implementations for each of the sources used in the mashups. Currently, the development of data mashups to deal with complex data structures requires strong programming skills, making mashups hard to create for novice users.

We utilize the concept of data mashups and use it to dynamically integrate heterogeneous web data sources by using the extension of XQuery proposed in [3]. All the available data sources over the internet are considered as a huge database and each data source is considered as a table. Data mashups can generate queries in extended XQuery syntax and can execute the sub-queries on any available data source contributing to the mashup. XML and RDF are the prevailing data formats for web data sources. To query these data sources, one can use XQuery and SPARQL – their respective query languages. The novelty of our tool is that it integrates the powerful features of database querying into a data mashup tool. It provides an easy to use interface of a mashup editor to generate complex queries visually for the integration of a multitude of distributed, autonomous, and heterogeneous data sources.

2 Database Oriented Mashups

A mashup application comprises three major components, which are (1) data level, (2) process level, and (3) presentation level [4]. The data level is mainly concerned with accessing and integrating heterogeneous web data sources. These sources can provide structured, semi-structured or unstructured data. Existing data mashup tools cannot deal with structural and semantic diversities of heterogeneous data sources. Recently, the importance of using data mashups for data integration using database oriented mashups has been realized [2]. Inspired by Yahoo pipes, there are a few attempts such as MashQL [5] and Deri Pipes [6] to generate semantic queries from data mashups. However, to the best of our knowledge, there exists no data mashup tool which allows the user to formulate queries over web data sources using their respective query languages and at the same time deals with the heterogeneity of the data sources. Our tool is similar to MashQL and Deri Pipes, but we focus on the XQuery extension of [3] with additional support of the SPARQL query language. Using our approach, existing data integration support for mashups is further enhanced to formulate a single query containing inside sub-queries of different query languages to deal with heterogeneous data integration.

⁴ <http://pipes.yahoo.com/pipes>

⁵ <http://code.google.com/gme>

⁶ www.ibm.com/software/info/mashup-center/

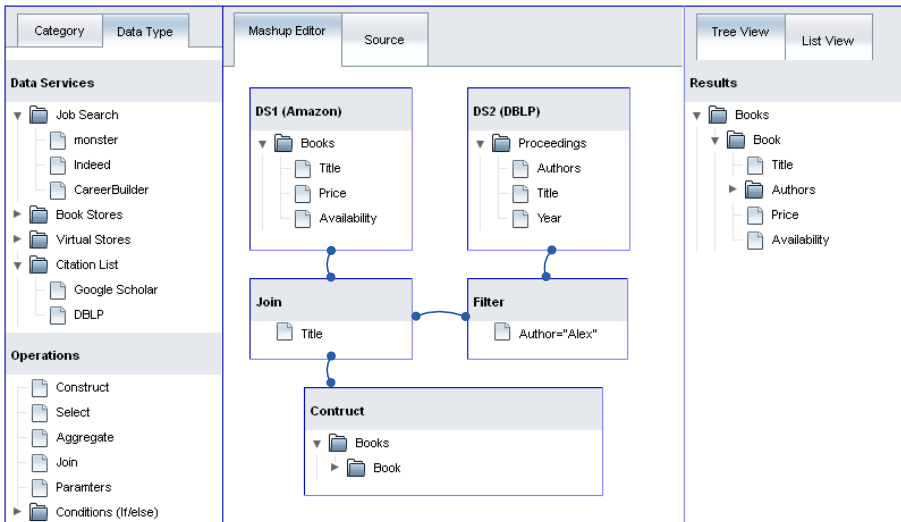


Fig. 1. Mashup Editor

3 Data Mashups Using XQuery

Figure 1 shows the interface of our system. The main window is divided into three panels, namely data source selection, mashup editor, and query results.

Data Source Selection. All available/registered data services are shown in the left panel of Figure 1. Each data service describes its available data source, its functionality and schema (if provided) which help the user to select the most suitable data service. Data services can be arranged in different categories based on metadata provided while registering a data source. Alternatively, data services can be grouped according to their data format (i.e., XML or RDF) by choosing the “data type” option in the left panel.


Mashup Editor. The central panel in Figure 1 is the mashup editor. The user can select any data service from the left panel and can easily drag and drop it into the mashup editor. These data services can be combined via several data operations, which are also selected by drag and drop from the left panel. The mashup is implemented by generating an extended XQuery expression with sub-queries in SPARQL or XQuery following the syntax in [3]. Figure 2 shows a sample extended XQuery expression generated by the mashup editor after integrating XML and RDF data sources. This query contains a SPARQL query as a sub-query inside XQuery. The mashup editor has both a design view (by choosing the “mashup editor” option in the central panel) and a command line interface (via the “source” option in the central panel). The design view provides an easy graphical interface while the command line interface is used by an expert user to write queries in the extended XQuery syntax described in [3]. For the creation of the queries from the graphical interface we use a similar approach as described in [7] for XQuery generation and [8] for SPARQL query generation.

Query Results. In the right panel of Figure 1, the result of executing the query from the mashup editor is displayed. The data format of the result is always XML. The user can


```

for $a in
doc("http://WISIRISFuzzySearch/License.xml")/agreement,
$b in SPARQLQuery(" SELECT ?Availability ?ExecutionTime
WHERE {
?x <http://www.w3.org/2001/sub#avail> ?Availability .
?x <http://www.w3.org/2001/sub#QoS> ?ExecutionTime "
} ,http://WISIRISFuzzySearch/QoS.rdf)/result
RETURN
  <Result>
    <ServiceTitle>{$a/title}</ServiceTitle>
    <Requirement>{$a/requirement}</Requirement>
    <Availability>{$b/availability}</Availability>
    <ExecutionTime>{$b/ExecutionTime}</ExecutionTime>
  </Result>

```

Fig. 2. A Sample query in extended XQuery – generated from the Mashup visual interface 

choose between two different views of the XML result: either in tree form (as shown in Figure 1) or in table form (by choosing the “list view” option in the right panel).

4 Conclusion

We provide a database oriented mashup tool for integrating heterogeneous data sources with a visual interface which allows for an easy definition of complex data mashups. This tool can be used as plug-in for web applications to generate powerful and efficient data integration mashups.

References

1. Dan, A., Johnson, R., Arsanjani, A.: Information as a service: Modeling and realization. In: Proc. SDSOA 2007. IEEE Computer Society, Los Alamitos (2007)
2. Vancea, A., Grossniklaus, M., Norrie, M.C.: Database-driven web mashups. In: Proc. ICWE, pp. 162–174. IEEE, Los Alamitos (2008)
3. Ali, M.I., Pichler, R., Truong, H.L., Dustdar, S.: DeXIN: An extensible framework for distributed XQuery over heterogeneous data sources. In: Filipe, J., Cordeiro, J. (eds.) ICEIS 2009. LNBI, vol. 24, pp. 172–183. Springer, Heidelberg (2009)
4. Lorenzo, G.D., Hacid, H., Paik, H.-Y., Benatallah, B.: Data integration in mashups. SIGMOD Record 38(1), 59–66 (2009)
5. Jarrar, M., Dikaiakos, M.D.: Querying the data web: The MashQL approach. IEEE Internet Computing 14(3), 58–67 (2010)
6. Morbidoni, C., Tummarello, G., Polleres, A.: Who the FOAF knows Alice? a needed step toward semantic web pipes. In: Proc. SWAP. CEUR Workshop Proceedings, vol. 314. CEUR-WS.org (2008)
7. Li Xiang, J.F.B., Gennari, J.H.: XGI: A graphical interface for XQuery creation. In: Proc. AMIA Symposium 2007, pp. 453–457. American Medical Informatics Association (2007)
8. Russell, A., Smart, P.R.: NITELIGHT: A graphical editor for SPARQL queries. In: International Semantic Web Conference (Posters & Demos). CEUR Workshop Proceedings, vol. 401. CEUR-WS.org (2008)

Executing In-network Queries Using SNEE

Ixent Galpin, Robert Taylor,
Alasdair J.G. Gray, Christian Y.A. Brennkmeijer,
Alvaro A.A. Fernandes, and Norman W. Paton

School of Computer Science, University of Manchester, UK
{ixent,a.gray,brennkmeijer,alvaro,norm}@cs.man.ac.uk
<http://snee.cs.manchester.ac.uk>

Keywords: Wireless Sensor Networks, In-network Processing, Distributed Query Processing, Stream Query Languages.

The SNEE query optimizer enables users to characterize data requests against *wireless sensor networks* (WSNs), using a declarative query language called SNEEqI (SNEE for **S**ensor **N**etwork **E**ngine, described in [GBG⁺11], and publicly available at <http://code.google.com/p/snee>). Queries are compiled into imperative *query execution plans*, which are translated into executable nesC source code¹. In this paper, we illustrate the lifecycle of a SNEEqI query Q for in-network execution. This lifecycle encompasses the steps of preparatory metadata collection, followed by the compilation of Q into a query execution plan QEP , the dissemination of binary images implementing QEP throughout the WSN, and the generation of query results.

To demonstrate our approach, we monitor light in a building using a simple 3-node WSN, depicted in Fig. 1, comprising of TelosB nodes². In our WSN, node 1 is the *gateway* node (i.e., the node from which commands and query execution plans are disseminated to the WSN, and also where query results are collected), and nodes 2 and 3 monitor light levels for the upstairs and downstairs areas of the building respectively. The schema comprises three logical streams, `building`, `upstairs` and `downstairs`, of type `(id:int, time:int, light:int)`. The `building` stream is the union of the `upstairs` and `downstairs` streams.

The example queries that we use to illustrate our approach are shown in Fig. 2. Query (a) requests all the light readings in the building; (b) requests the average value of the light readings in the building; and (c) requests light readings when the light level upstairs is higher than downstairs (i.e., it may indicate that someone has forgotten to switch off a light). The QoS expectations are both an *acquisition interval* and *delivery time* of 10s (i.e., query results need to be delivered before the next tuple is acquired).

¹ See <http://www.tinyos.net>.

² This hardware has the following specification: CPU = MSP430 8MHz, RAM = 10 kB, Program Memory = 48 kB, Data Flash = 1 MB, Radio = CC2420. Detailed specifications can be found at http://www.willow.co.uk/TelosB_Datasheet.pdf.

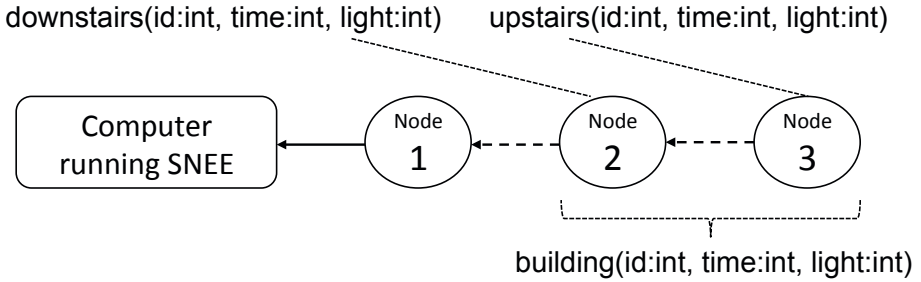


Fig. 1. Network assumed and sources for each stream

```

SELECT  RSTREAM id, light      SELECT  RSTREAM avg(light)
FROM    building[NOW];        FROM    building[NOW];

(a) All light readings        (b) Average light readings

SELECT  RSTREAM u.time, u.light, d.light
FROM    upstairs[NOW] u, downstairs[NOW] d
WHERE   u.light > d.light;

(c) Correlating Light Readings with Filtering Condition
    
```

Fig. 2. Example SNEEql Queries

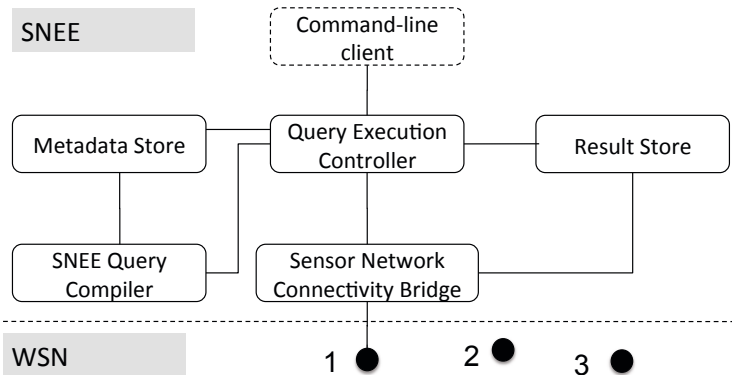


Fig. 3. SNEE components

Fig. 3 presents the interactions between components of the system. The *Sensor Network Connectivity Bridge* is responsible for interfacing with the WSN. On WSN nodes, a boot loader is used to switch between 3 program images in flash memory, depicted in Fig. 4. This is necessary due to the limited size of the program memory. Images in slots 1 and 2 are preinstalled prior to WSN

Metadata Collector	1
Over-the-Air Programmer	2
Query Execution Plan	3

Fig. 4. Node Images in Flash Memory

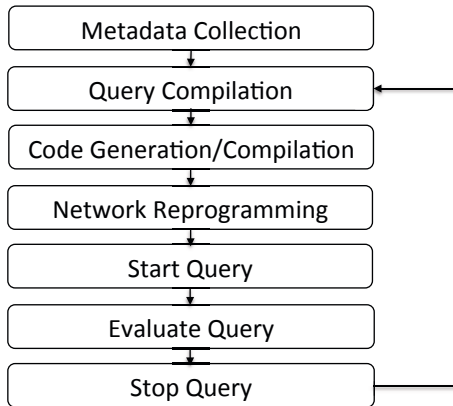


Fig. 5. Lifecycle Steps

deployment. The lifecycle of a query involves the steps depicted in Fig. 5 and described below:

Metadata Collection. Nodes run the *Metadata Collector* program which generates a description of the WSN topology (e.g., expected energy transmission energy between nodes, based on average package loss) for use by the SNEE compiler. This program is based on the *Collection Tree Protocol*³, and works by sending routing table entries of each node to the gateway. When this completes, the nodes reboot into the *Over-the-Air Programmer* (OTAP).

Query Compilation. The user poses a query and Quality-of-Service expectations, which is compiled using the *SNEE Query Compiler* into a query execution plan based on the metadata that was collected in the previous step. Query compilation is described in detail in [GBG⁺11].

Code Generation/Compilation. The Sensor Network Connectivity Bridge translates the query execution plan into a nesC program for each node, each of which is compiled into a binary image.

³ This is an implementation of the tree formation protocol described in <http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html>.

Network Reprogramming. The OTAP is used to transmit an executable image to slot 3 of each node. The OTAP is based on Deluge [HC04] but has been extended to enable different executable images to be sent to each node in the WSN. This is necessary because SNEE generates query execution plans with distinct tasks for each node in the WSN, rather than sending the same query execution plan to nodes in the WSN. This enables SNEE to be more economical with respect to memory consumption⁴.

Start Query. Nodes reboot into the query execution plan, which starts running when the Sensor Network Connectivity Bridge issues a *start* command. Commands are disseminated using Drip [TC05] a mechanism used to propagate small values across all nodes in a WSN⁵.

Evaluate Query. The nodes evaluate the query execution plan, sending results back to gateway node.

Stop Query. Query results are produced until a *stop* command is issued.

This paper has outlined the lifecycle for SNEE queries. An experimental evaluation measuring metrics including energy consumption and lifetime of SNEE queries, using simulation and real hardware, is presented in [GBG⁺11]. The development of a query compiler/evaluator for WSNs leaves open a collection of query lifecycle management issues; this paper describes an approach in which support tasks such as OTAP and metadata collection are represented as separate programs that work together with query executables to support declarative, energy efficient environment monitoring.

Acknowledgements. This work is part of the SemSorGrid4Env project funded by the European Commission's Seventh Framework Programme.

References

- [GBG⁺11] Galpin, I., Brenninkmeijer, C.Y.A., Gray, A.J.G., Jabeen, F., Fernandes, A.A.A., Paton, N.W.: SNEE: a query processor for wireless sensor networks. DAPD 29(1-2), 31–85 (2011)
- [HC04] Hui, J.W., Culler, D.E.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: SenSys, pp. 81–94 (2004)
- [TC05] Tolle, G., Culler, D.E.: Design of an application-cooperative management system for wireless sensor networks. In: EWSN, pp. 121–132 (2005)

⁴ See [GBG⁺11] for memory consumption figures of SNEE for the query execution plan image, excluding management functionality.

⁵ This is an implementation of the dissemination framework described in <http://www.tinyos.net/tinyos-2.x/doc/html/tep118.html>.

Extracting Data Records from Query Result Pages Based on Visual Features

Daiyue Weng, Jun Hong, and David A. Bell

School of Electronics, Electrical Engineering and Computer Science,
Queen's University Belfast, Belfast BT7 1NN, UK
{dweng01, j.hong, da.bell}@qub.ac.uk

Abstract. Web databases contain a large amount of structured data which are accessible via their query interfaces only. Query results are presented in dynamically generated web pages, usually in the form of data records, for human use. The problem of automatically extracting data records from query result pages is critical for web data integration applications, such as comparison shopping sites, meta-search engines, etc. A number of approaches to query result extraction have been proposed. As the structures of web pages become more complex, these approaches start to fail. Query result pages usually also contain other types of information in addition to query results, e.g., advertisements, navigation bar, etc. Most of the existing approaches do not remove such irrelevant contents which may affect the accuracy of data record extraction. We have observed that query results are usually displayed in regular visual patterns and terms used in a query often re-appear in query results. We propose a novel approach that makes use of visual features and query terms to identify the data section and extract data records from it. We also use several content and visual features of visual blocks in a data section to filter out noisy blocks. The results of our experiments on a large set of query result pages in different domains show that our proposed approach is highly effective.

1 Introduction

The volume of structured data on the Web has been increasing enormously. Such data are usually returned from back-end databases in response to specific user queries, and presented in the form of data records in query result pages. Access to web databases is via their query interfaces (usually HTML query forms) only. In literature, the contents of web databases are usually referred to as the Deep Web. A recent study [20] estimates that the number of web databases that are 'hidden' on the Web is well in the order of 10^5 and continues expanding rapidly. Many e-commerce sites are supported by web databases.

In general, the majority of query result pages are list pages, each of which contains a number of data records in columns with each row on each column representing a data record. For example, Figure 1 shows a list page from cooking.com, which has a single column containing 10 data records about plates.

The image shows a screenshot of the Cooking.com website's search results page for the query "Accent Plates". The page features a navigation bar with categories like SHOP, VALUES, GIFTS, CELEBRITY, RECIPES, and COMMUNITY. A sidebar on the left contains various shopping navigation options such as "Holiday Gift Guide", "Free Shipping Offer", and "Best Brands". The main content area displays a list of products, each with a thumbnail image, a title, a description, and a price. The products include items from brands like Denonware, Lenox, Mikasa, and Royal Doulton. At the bottom of the page, there is a section for "Begin a new search for" and a "SUBMIT" button.

Image	Product Name	Description	Price
	Accent Plates by Denonware, Barcelona	9-in. Accent Plate	\$34.95
	Accent Plates by Denonware, Nightcap	9-in. Accent Plate Black	\$50.95
	Accent Plates by Denonware, Phoenix Platinum	9-in. Accent Plate Evergreen (1 rating) 9-in. Accent Plate Peridot (1 rating) 9-in. Accent Plate Apricot (1 rating) 9-in. Accent Plate Black (1 rating) 9-in. Accent Plate Cocoa (1 rating) 9-in. Accent Plate Raspberry (1 rating)	\$39.95 \$39.95 \$44.95 \$44.95 \$44.95 \$44.95
	Accent Plates by Denonware, Serenity	9.5-in. Accent Plate White	\$22.95
	Accent Plates by Denonware, Fenyson	9-in. Accent Plate	\$59.95
	Accent Plates by Denonware, Fenyson	9-in. Holiday Accent Plate	\$59.95
	Dinner Plates by Denonware, Provence Noir	11-in. Plaid Dinner Plate	\$21.95
	Salad Plates by Denonware, Courtesa	9-in. Accent Salad Plate	\$34.95
	Salad Plates by Denonware, Provence Noir	9-in. Dark Salad Plate	\$14.95
	Salad Plates by Denonware, Provence Noir	9-in. Plaid Salad Plate	\$14.95

Fig. 1. An example query result page

Extracting data records from query result pages enables integrating data from a magnitude of web databases to generate value-added web applications, such as price comparison sites and meta-search engines, etc. Query result pages are dynamically generated from back-end databases in response to user queries and encoded in HTML using pre-defined templates or script programs. These pages are semi-structured and displayed for human use, rather than for processing by programs. How to automatically extract data records into a structured form that is machine processable is a very challenging problem.

There has been a lot of research on fully-automatic approaches [3–16] for extracting data from query result pages. Those in [3–10] represent the current technical trend of query result extraction. First, they identify a data section, which contains a set of data records. Second, they identify data records from the data section. Finally, they extract data by aligning the corresponding attributes of different records, producing a relational table [4, 5, 8, 10].

However, the existing approaches to query result extraction have some inherent limitations. First, web pages are becoming more complex; their tag structures are ever-growing complex since HTML itself is evolving constantly, and other technologies like JavaScript and CSS are widely deployed to make result pages more dynamic. This may make the layouts of result pages different from their tag tree or token string representations, and thus the existing approaches that rely on such representations may fail. Second, some of the existing approaches employ a similarity measure on page segments to identify data records. However, data records may not be extracted correctly if the sibling tree segments of the same root are not similar to each other. This also makes it impossible to extract a single data record in the data section. Third, most of the existing approaches do not filter out noisy contents. Noisy contents refer to any parts of a query result page that are not part of any data record, e.g. banner advertisements, navigation bar, copyright notice, record statistical information etc. We are most interested in the part of a result page which contains all the data records with few noisy contents which often affect the accuracy of data record extraction. Thus it is very important to remove any noisy contents before data record extraction.

In this paper, we focus on the problem of data record extraction, that is, given a query result page that contains a single column of data records, automatically identify the data section and data records. We propose a novel approach to overcome the limitations of the existing approaches. First, our approach transforms a query result page into a Visual Block tree using the VIPS algorithm [17], which represents a visual partition of the web page. Such a representation reflects the content structure of the page enforced by visual cues so that content related data items are represented in the same branch of the Visual Block tree. For example, Figure 2 shows a visual partition of the result page shown in Figure 1. Figure 3 shows part of the visual block tree for visual block $b_{1-1.2.2.2.3.4}$. We can also get visual features (e.g., positions, width, height etc) of each block on the Visual Block tree.

Second, our approach identifies the data section by exploiting the sizes of the visual blocks of the result page, and counting the occurrences of query terms

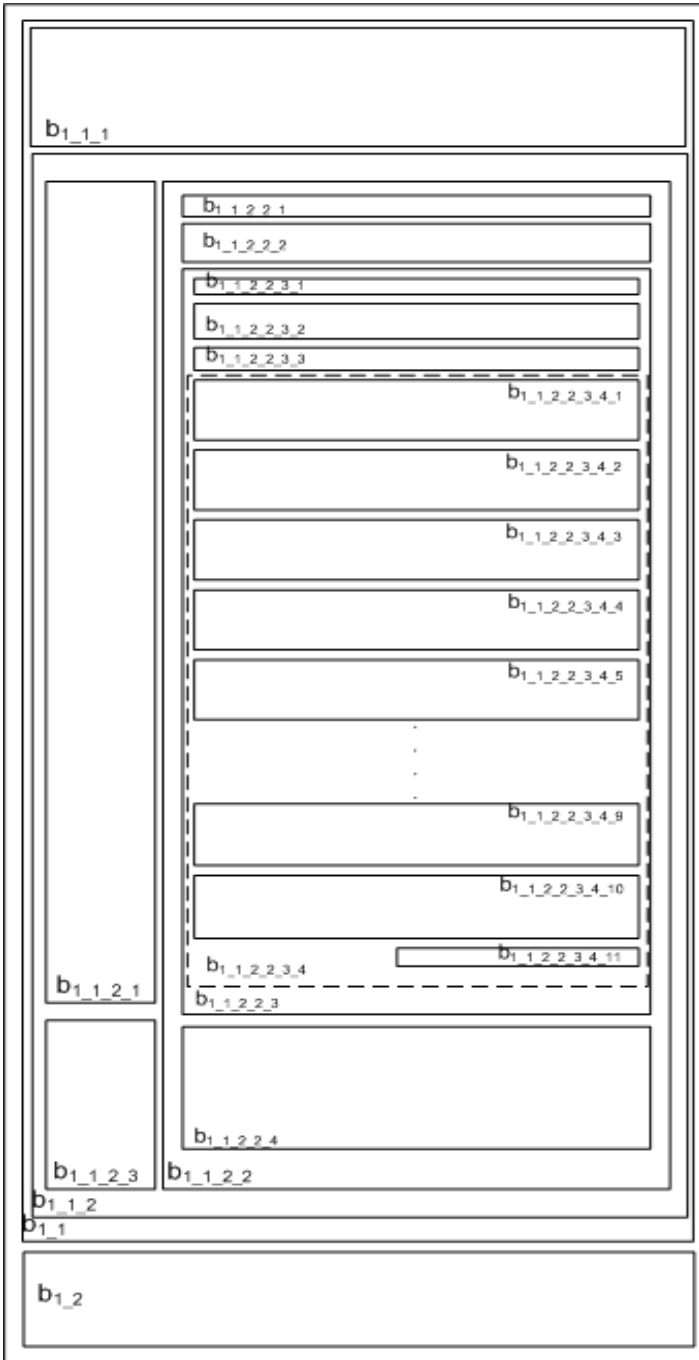


Fig. 2. The visual block layout of the query result page shown in Fig. 1

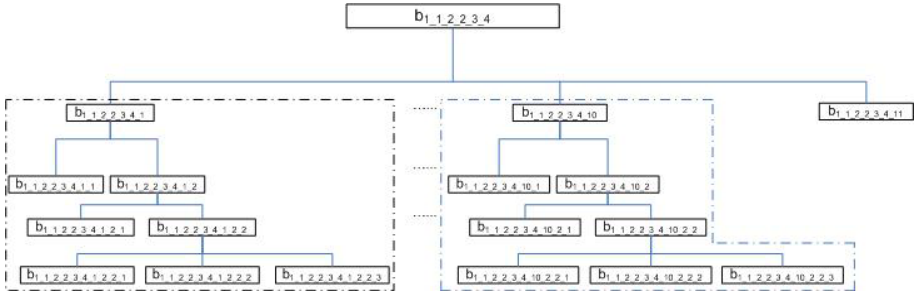


Fig. 3. Part of the Visual Block tree for $b_{1.1.2.2.3.4}$

in them. This is based on the following two observations: the data section in a result page, which contains all the data records, usually occupies a significant area of the result page; query terms often re-appear in the data records. We use these two observations to identify the data section. For example, block $b_{1.1.2.2.3.4}$ in Figure 2 is likely to be the data section since it occupies a large portion of the result page in Figure 1 and contains a number of occurrences of query terms (e.g., “Accent Plates”). To get query terms, we make use of the query interface and assume that the result pages are generated in response to the queries made via the interface.

The identified data section often contains noisy blocks. Data records are obviously more vivid in content than noisy blocks, have one or more links or some images. To filter out noisy blocks, we use a vector of content and visual features to characterize each block within the data section. These features provide statistical information about texts, block area, links and images in the block. The overall importance of a block for a data record should be higher than noisy blocks. We set up a threshold of importance to ensure that any blocks that have less importance than the threshold are identified as noisy blocks and removed. For example, as shown in Figure 1 there are ten data record blocks while the block containing information about the data records (“Items (1 - 15) of 15”) is identified as a noisy block and removed.

Third, we observe that each data record contains semantically related data units of a data object, which reside in the leaf nodes of the Visual Block trees, and are visually aligned with and adjacent to each other. Our approach identifies data records by purely using the rendering boxes of the leaf nodes in the data section to infer their alignment and proximity. For example, the data units of each data record shown in Figure 1 are aligned with each other, in close proximity and relatively far away from the data units of the other data records. Thus we can group data units based on their positional information with each group representing a data record.

In summary, we make the following contributions. First, we propose an approach for identifying data sections based on the visual features of the blocks and re-occurrences of query terms in them. Based on the content and visual features of visual blocks, our approach for removing noisy blocks can eliminate most of

the noisy blocks. Second, we propose an approach for identifying data records based on an observation that the data units of a data record are visually aligned with and close to each other, and that they are distant from the data units of the other data records. By grouping data units in such a way, our approach does not miss any data record that is not similar to the other data records, and our approach can extract a single data record from a query result page.

The rest of this paper is organized as follows. Section 2 presents web page representation, the problem definition and an overview of our approach. Sections 3 - 5 describe our approaches for identifying data sections, removing noisy blocks and identifying data records. Experimental results are given in section 6. Section 7 discusses related work. Section 8 concludes the paper.

2 Fundamentals and Overview

In this section, we first introduce Visual Block trees and give a formal definition of the rendering box model of web pages based on the Visual Block tree, which is the basis of our approach. We then define the problem of data record extraction and present an overview of our approach.

2.1 Visual Representation of Query Result Pages

The content of a query result page is typically organized into different regions to make it easy for human use, e.g., advertisements, menu bar, sponsor links, query results and so on. Each region contains semantically related content. Visual cues (e.g. lines, spaces, font sizes, background colours etc) can be used to distinguish regions from each other. To make use of visual features for data record extraction, we employ the VIPS [17] algorithm to represent a query result page as a Visual Block tree. The root of the tree represents the entire page and each node represents a rendering box (a visual block) on the page. A leaf node represents a block containing a basic semantic unit that cannot be further decomposed, e.g., a text or image. Node a is an ancestor of node b if the block that a represents contains the block that b represents on the page. The blocks represented by nodes at the same level of the tree do not overlap. The order of the child nodes with the same parent follows the order of the blocks they represent on the page, i.e., top-down, left-right. For example, Figure 2 shows the visual block layout produced by the VIPS algorithm for the query result page shown in Figure 1. For example, b_1 represents the body of the page, $b_{1.1.2.1}$ represents the block containing the category links on the page, $b_{1.2}$ contains the website information and $b_{1.1.2.2.3.4}$ contains all data records denoted as $b_{1.1.2.2.3.4.1}$ to $b_{1.1.2.2.3.4.10}$. Figure 3 shows part of the Visual Block tree for $b_{1.1.2.2.3.4}$.

2.2 Overview of Our Approach

Given the Visual Block tree of a query result page, first we identify a visual block that contains all the data records and treat it as the data section. Second, we

remove any noisy blocks in the data section. Third, we group leaf nodes of the Visual Block tree into data records based on the positions of their corresponding visual blocks. The approach takes as input a query result page from a specific web database, and produces as output a set of data records.

3 Identifying Data Sections

We identify a *data section* as a node in the Visual Block tree, which represents a rectangular box in the result page that contains all the data record blocks and as few noisy blocks as possible.

We observe that the size of a data section is usually large relative to the size of the whole page. For example, as shown in Figure 11, the data section that contains all the plate products occupies a relatively large area. To utilize the observation, we first select those blocks, each of which satisfies a constraint that the ratio between the sizes of the block and the whole page is greater than a threshold T_{dr} (16), which can be trained from sample result pages.

The method for identifying data sections first takes the root node of the Visual Block tree as input. It returns a set of candidate data section blocks. The blocks at higher levels of the Visual Block tree occupy bigger portions of the result page so that their area ratios are much higher than the threshold and will certainly contain more noisy blocks than the ones at lower levels of the Visual Block tree. The algorithm selects candidate data section blocks in a depth-first fashion. It traverses the Visual Block tree from the root, and identifies those blocks that satisfy the area ratio constraint but none of their child blocks changes it that. These blocks thus contain less noisy blocks. For example, after applying the area constraint, we can identify $b_{1,1}$, $b_{1,1,2}$, $b_{1,1,2,2}$, $b_{1,1,2,2,3}$ and $b_{1,1,2,2,3,4}$ as candidate data sections.

Candidate data sections are further considered to determine the real data section. To do this we make use of query terms that are used in queries over query interfaces. A query interface exposes the attributes of the web database schema to the user and usually consists of a set of input elements, e.g., text boxes, radio buttons, check boxes and selection lists. Each input element is associated with an attribute (18). For example “Dinnerware” “Plates” “Royal Doulton” and “\$25 to \$50” are query terms used for input elements associated with attributes “Category” “Product type” “Brand” and “Price” of the query interface, as shown in Figure 4. We observe that query terms often re-appear in the data records. For example, the data records shown in Figure 11 are in response to the query shown in Figure 4. We can see that the text nodes of each data record contain the occurrences of query terms “Plates” and “Royal Doulton”.

The frequency of each query term in a candidate block reflects the importance of the candidate block. The more query terms occur in a block, the more likely the block is the data section. Given a set of query terms q_i for $i = 1, 2, \dots, n$, and a candidate block, the importance of the block is measured as $R = \sum_{i=1}^n f_i$, where f_i represents the frequency of query term i in the candidate block. The block that has the maximum number of occurrences of query terms among all the

PRODUCT SEARCH

To Begin Use **one or more** of the following criteria

Category We recommend you choose one category to narrow down your search results

Dinnerware e.g. Bakeware

To further narrow your search results, choose a product type:

Plates e.g. Muffin Pans

Brand Royal Doulton

Price \$25 to \$50

Finished?

Fig. 4. The query interface of cooking.com

candidate blocks is identified as the data section. For example, after applying the second constraint to the candidate data sections, $b_{1-1.2.2-3.4}$, as shown in Figure 3 is identified as the data section .

4 Removing Noisy Blocks

The identified data section usually contains noisy blocks on the top and bottom of the section, and data records in the middle of the section with no noisy blocks on either the left or right of the records. Noisy blocks are the ones that are in a data section but are not part of any data record [16], such as data record numbers (e.g., “Items (1-15) of 15” in Figure 1). We observe that a data record typically contains images, description of data, links, and occupies a significant area on the page. For example, each of the data records shown in Figure 1 contains the image, name, and model etc of the product, one or more links for detailed information about a specific model and the rectangle of each data record is very noticeable. Specifically, we evaluate the importance of each first-level child block within the data section by using the five features about the content of the block: $ImgNum$ (the number of images in the block), $LinkNum$ (the number of links in the block), $LinkTextLen$ (the anchor text length of the block), $TextLen$ (the text length of the block), and $Area$ (the rendering area of the block).

These content features are provided by the Visual Block tree and are normalized across the whole data section block. The importance of a child block is defined as $ImBlk = w_1 \times ImgNum + w_2 \times LinkNum + w_3 \times LinkTextLen + w_4 \times$

$TextLen + w_5 \times Area$, where w_1, w_2, w_3, w_4 and w_5 are real numbers so that $w_1 + w_2 + w_3 + w_4 + w_5 = 1$, and $0 \leq ImBlk \leq 1$. $LinkTextLen$ and $TextLen$ are considered as the most important features for differentiating data record blocks from noisy blocks. When the $ImBlk$ of a block is greater than the given threshold θ , it is very likely that the block is a data record. Otherwise the block is taken as a noisy block. The threshold can be trained using sample pages.

5 Grouping Data Units of Data Records

A data record represents a data object retrieved from a web database and consists of multiple data units that are semantically related. Data units are represented as leaf nodes on the Visual Block tree, and they are visually aligned with and adjacent to each other on query result pages. For example, as shown in Figure 1, the data units of each record are the leaf nodes in the Visual Block tree, and they are visually aligned with and adjacent to each other on the web page. To identify data records, our approach first identifies leaf nodes that are part of a data record and can be used as starting points for grouping other data units of the record. Given a starting point, our approach first group data units that are horizontally aligned with it to form a data unit group based on the positions of the visual blocks of the corresponding leaf nodes. It then groups data units that are horizontally aligned with each other to form leaf node groups. Finally, our approach progressively expands each data unit group with other data unit groups and leaf node groups that are vertically adjacent to it until there is no vertically adjacent group. Each data unit group thus corresponds to a data record.

Definition 1. (*Block and group positions*) - We use the coordinate of the top-left corner, height and width of the visual block of a data unit to determine its left, right, top and bottom positions. Furthermore, we use the left position of the leftmost node of a node group as the left position of the group, the top position of the topmost node as the top position of the group, the right position of the rightmost node of a node group as the right position of the group, and the bottom position of the bottom node as the bottom of the group.

Definition 2. (*Horizontal alignment*) - We say that two leaf nodes, a and b , are horizontally aligned with each other, if they have similar top positions. Furthermore, we say that two node groups, a and b , are horizontally aligned with each other, if they have similar top positions.

Definition 3. (*Vertical adjacency*) - We say that two leaf nodes, a and b , are vertically adjacent, if the distance between the bottom position of a and the top position of b , or the distance between the top position of a and the bottom position of b (vertical distance) is less than a given number of pixels (in close proximity). Furthermore, we say that two node groups, a and b , are vertically adjacent, if the shortest vertical distance between the nodes in a and b is less than a given number of pixels (in close proximity), and the nodes in the two groups are on the same subtree.

The algorithm, as shown in Algorithm 1, takes as input a set of query terms (denoted as T) and a data section block (denoted as B), and identifies as output a set of data records (denoted as R). The algorithm first identifies starting leaf nodes (lines 2-6) by matching each query term with each text node in the Visual Block tree of the data section. Second, the algorithm forms a data unit group with each starting leaf node and tries to expand it with leaf nodes that are horizontally aligned with it (lines 7-13). Third, the algorithm groups leaf nodes that are not included in data unit groups, and horizontally aligned with each other, into leaf node groups G_l (lines 14-22). Fourth, the algorithm expands each data unit group with the other data unit group that are horizontally aligned with it (lines 23-30). Fifth, the algorithm expands each data unit group with other data unit group and leaf node groups that are vertically adjacent to it (lines 31-42). The algorithm ends when there is no more leaf node group or data unit group that is vertically adjacent to the expanding data unit group.

To illustrate how the algorithm works, we take the first two data records shown in Figure 1 as an example. “Plates” has been used as a query term. Two leaf nodes representing text “Dinner Plates by” are identified as starting leaf nodes. These two starting leaf nodes are used to initiate two data unit groups. Those leaf nodes representing the second rows of these two data records form two leaf node groups. Each data unit group is expanded with a leaf node group which is vertically adjacent to it. Each extracted data unit group represents a data record.

6 Experimental Results

We have implemented a prototype in Visual C++ based on our proposed approach. Every query result page is first parsed by the VIPS into a Visual Block tree which the prototype takes as input. We have conducted experiments on a data set of 200 query result pages that are returned from 20 web databases in the UIUC Web Integration Repository [19]. These web databases are from 5 domains - Books, Jobs, Movies, Music and Hotels. 15 of these pages contain a single data record. For each web database, 10 result pages are collected after manually submitting 10 different queries via its query interface. We use two common measures, *recall* and *precision*, to evaluate the performance of our approach. Recall is the percentage of the number of data records that have been correctly extracted over the total number of data records on a result page. Precision is the percentage of the number of data records that have been correctly extracted over the total number of data records that have been extracted.

We compare our approach with MDR [3], which is a well known data record extraction system. We set the similarity threshold for MDR at its recommended value (60%). Table 1 shows the experimental results of both our approach and MDR. As we can see from Table 1, our approach has much better experimental results than MDR in total, and in almost every domain our approach significantly outperforms MDR. The precision and recall of our approach are both high across all domains, approaching 100%. Our approach can also extract query result pages

Algorithm 1. Grouping Data Units

Input: a set of query terms T , a data section block B **Output:** a set of data records R

```

1: Set  $R$ , a set of leaf nodes  $N_l$ , a set of starting leaf nodes  $N_s$ , a set of data unit
   groups  $G$ , a set of leaf node groups  $G_l$ , and a set of horizontally expanded data
   unit groups  $G'$  all to  $\{\}$ 
2: Add every text node in  $B$  to  $N_l$ 
3: for every leaf node  $n_l \in N_l$  do
4:   if  $n_l$  contains a query term  $t \in T$  then
5:     Add  $n_l$  to  $N_s$ 
6:     Remove  $n_l$  from  $N_l$ 
7: for every starting leaf node  $n_s \in N_s$  do
8:   Set a data unit group  $g$  to  $\{n_s\}$ 
9:   for every leaf node  $n_l \in N_l$  do
10:    if  $n_l$  is horizontally aligned with  $n_s$  then
11:      Add  $n_l$  to  $g$ 
12:      Remove  $n_l$  from  $N_l$ 
13:    Add  $g$  to  $G$ 
14: repeat
15:   Remove a leaf node  $n_l$  from  $N_l$ 
16:   Set a leaf node group  $g_l = \{n_l\}$ 
17:   for each leaf node  $n'_l \in N_l$  do
18:     if  $n'_l$  is horizontally aligned with  $n_l$  then
19:       Add  $n'_l$  to  $g_l$ 
20:       Remove  $n'_l$  from  $N_l$ 
21:   Add  $g_l$  to  $G_l$ 
22: until  $N_l = \{\}$ 
23: repeat
24:   Remove a data unit group  $g$  from  $G$ 
25:   for each data unit group  $g' \in G$  do
26:     if  $g'$  is horizontally aligned with  $g$  then
27:       Set  $g$  to  $g \cup g'$ 
28:       Remove  $g'$  from  $G$ 
29:   Add  $g$  to  $G'$ 
30: until  $G = \{\}$ 
31: repeat
32:   Remove a horizontally expanded data unit group  $g'$  from  $G'$ 
33:   for each horizontally expanded data unit group  $g'' \in G'$  do
34:     if  $g''$  is vertically adjacent to  $g'$  then
35:       Set  $g'$  to  $g' \cup g''$ 
36:       Remove  $g''$  from  $G'$ 
37:   for each leaf node group  $g_l \in G_l$  do
38:     if  $g_l$  is vertically adjacent to  $g'$  then
39:       Set  $g'$  to  $g' \cup g_l$ 
40:       Remove  $g_l$  from  $G_l$ 
41:   Add  $g'$  to  $R$ 
42: until  $G' = \{\}$ 
43: Return  $R$ 

```

with single data records, but MDR cannot. Table 1 shows that our approach has slightly higher precision than recall. The main reasons for missing data records are as follows. First, sometimes some data records do not contain any query terms so our approach cannot identify the appropriate starting leaf nodes. Second, sometimes the VIPS divides a data section into multiple sections, and our approach only identifies the largest section as the data section. The main reasons for extracting data records incorrectly are as follows. First, some noisy blocks have not been removed from the data section because they may contain query terms. Second, sometimes the VIPS parses result pages incorrectly so that some data items are missing on the Visual Block tree. Third, sometimes the VIPS fails to give correct block positions, which leads to data units missing from some data records. The performance of MDR is inversely proportional to the complexity of the result pages, and it performs relatively well on extracting data records from tables.

Table 1. Comparison results between our approach and MDR

Domain	Our Approach		MDR	
	Precision	Recall	Precision	Recall
Books	97.86%	96.76%	40.38%	82.01%
Hotel	99.20%	98.30%	18.21%	32.68%
Jobs	99.48%	98.37%	99.62%	67.60%
Movies&Music	100%	98.54%	28.05%	72.46%
Single Record Page	100%	100%	0%	0%
Total	99.26%	98.11%	38.68%	74.86%

7 Related Work

Automatic extraction of web query results has attracted a lot of attention over the recent years. Several automatic extraction systems have been developed. Earlier works mainly focus on finding repetitive patterns and templates in result pages, e.g., IEPAD [13], RoadRunner [12], DeLa [14] and EXALG [15]. Recent techniques have focused on exploiting tag structures and visual features, e.g., MDR [3], DEPTA [4, 5], MSE [7], ViNTs [6], ViPER [8], ViDE [16] and [9].

The works that use visual features include ViPER [8], ViNTs [6], MSE [7] and ViDE [16]. ViDE, is the most related to our approach. It is the first work that is primarily based on visual features. There are several main differences between ViDE and our approach. ViDE first clusters data units of the same semantics based on similarity between their appearances, and then groups appropriate data units from each of the clusters into data records. Our approach uses a proximity-based technique to directly group data units in the same data records. ViDE may cluster data units with different semantics because sometimes neighboring data units in the same data record may not have distinguishable appearances, resulting in them being clustered together and then grouped into different data records. Second, ViDE uses the positions and sizes of visual blocks to determine if a block is the data section. If multiple blocks are identified as candidate data

sections, it chooses the one with smallest size as the data section. Our approach counts the occurrences of query terms in candidate blocks to select the real data section that makes our approach more robust. Third, ViDE identifies noisy blocks by deciding whether the blocks are aligned to the left of a data section but it may not remove all the noisy blocks. Our approach evaluates the importance of blocks within the section based on content and visual features which improve the effect of removing noisy blocks.

Our algorithm for grouping data units of a data record is inspired by the work of Gatterbauer and Bohunsky [1, 2] on extracting web tables. Our approach instead extracts data records from query result pages that have more complex content structures. Though our approach also uses the alignment and adjacency techniques, our alignment definition is much simpler than the one in [1, 2]. Our approach uses also query terms in the process of grouping data units.

8 Conclusions

In this paper, we present an automatic approach for extracting data records from query result pages. Our approach first uses the sizes of visual blocks and the occurrences of query terms in visual blocks to identify the data section. It then groups data units in the data section, which are in close proximity, into data records. It also uses content and visual features of visual blocks to evaluate their importance and to filter out noisy blocks. Our work can be part of a web data integration system which interacts with multiple web databases, e.g. e-commerce web sites. Our experimental results show that our proposed approach is highly effective. In future work, we will develop algorithms for aligning data units in the extracted data records so that data units of the same attribute can be aligned into the same column of the query result table.

References

1. Gatterbauer, W., Bohunsky, P., Herzog, M., Krupl, B., Pollak, B.: Towards Domain-Independent Information Extraction from Web Tables. In: WWW 2007, pp. 71–80 (2007)
2. Gatterbauer, W., Bohunsky, P.: Table Extraction Using Spatial Reasoning on the CSS2 Visual Box Model. In: AAAI 2006, pp. 1313–1318 (2006)
3. Liu, B., Grossman, R., Zhai, Y.: Mining Data Records in Web Pages. In: KDD 2003, pp. 601–606 (2003)
4. Zhai, Y., Liu, B.: Web Data Extraction Based on Partial Tree Alignment. In: WWW 2005, pp. 76–85 (2005)
5. Zhai, Y., Liu, B.: Structured Data Extraction from the Web Based on Partial Tree Alignment. *IEEE Trans. on Knowl. and Data Eng.* 18(12), 1614–1628 (2006)
6. Zhao, H., Meng, W., Wu, Z., Raghavan, V., Yu, C.: Fully automatic wrapper generation for search engines. In: WWW 2005, pp. 66–75 (2005)
7. Zhao, H., Meng, W., Yu, C.: Automatic Extraction of Dynamic Record Sections from Search Engine Result Pages. In: VLDB 2006, pp. 989–1000 (2006)

8. Simon, K., Lausen, G.: ViPER: Augmenting Automatic Information Extraction with Visual Perceptions. In: CIKM 2005, pp. 381–388 (2005)
9. Miao, G., Tatemura, J., Hsiung, W., Sawires, A., Moser, L.E.: Extracting Data Records from the Web Using Tag Path Clustering. In: WWW 2009, pp. 981–990 (2009)
10. Liu, B., Zhai, Y.: NET - A System for Extracting Web Data from Flat and Nested Data Records. In: Ngu, A.H.H., Kitsuregawa, M., Neuhold, E.J., Chung, J.-Y., Sheng, Q.Z. (eds.) WISE 2005. LNCS, vol. 3806, pp. 487–495. Springer, Heidelberg (2005)
11. Zhu, J., Nie, Z., Wen, J., Zhang, B., Ma, W.: Simultaneous Record Detection and Attribute Labeling in Web Data Extraction. In: KDD 2006, pp. 494–503 (2006)
12. Crescenzi, V., Mecca, G., Merialdo, P.: RoadRunner: Towards Automatic Data Extraction from Large Web Sites. In: VLDB 2001, pp. 109–118 (2001)
13. Chang, C.-H., Lui, S.-C.: IEPAD: Information Extraction Based on Pattern Discovery. In: 10th International Conference on World Wide Web, pp. 681–688. ACM, New York (2001)
14. Wang, J., Lochovsky, F.H.: Data Extraction and Label Assignment for Web Databases. In: WWW 2003, pp. 187–196 (2003)
15. Arasu, A., Garcia-Molina, H.: Extracting Structured Data from Web Pages. In: SIGMOD 2003, pp. 337–348 (2003)
16. Liu, W., Meng, X.F., Meng, W.Y.: ViDE: A Vision-Based Approach for Deep Web Data Extraction. *IEEE Trans. on Knowl. and Data Eng.* 22(3), 447–460 (2010)
17. Cai, D., Yu, S., Wen, J., Ma, W.: Extracting Content Structure for Web Pages Based on Visual Representation. In: Zhou, X., Zhang, Y., Orłowska, M.E. (eds.) APWeb 2003. LNCS, vol. 2642, pp. 406–417. Springer, Heidelberg (2003)
18. Wang, J., Wen, J., Lochovsky, F., Ma, W.: Instance-based schema matching for web databases by domain-specific query probing. In: VLDB 2004, pp. 408–419 (2004)
19. The UIUC Web Integration Repository, <http://metaquerier.cs.uiuc.edu/repository/>
20. Madhavan, J., Jeffery, S.R., Cohen, S., Dong, X.L., Ko, D., Yu, C., Halevy, A.: Web-scale Data Integration: You Can Only Aford to Pay as You Go. In: CIDR 2007, pp. 342–350 (2007)

Updates on Grammar-Compressed XML Data

Alexander Bätz, Stefan Böttcher, and Rita Hartel

University of Paderborn, Computer Science, Fürstenallee 11, 33102 Paderborn, Germany
{laures, stb, rst}@uni-paderborn.de

Abstract. In this paper, we present updates on CluX, a grammar-based XML compression approach based on clustering XML sub-trees. We show that updates on CluX-compressed data can be performed faster than decompressing the data, loading it into main memory and compressing it. Furthermore, we show how to support fast multiple updates, e.g. performing 100 updates in parallel is more than 70 times faster than 100 single updates.

Keywords: updating compressed XML data, grammar-based compression.

1 Introduction

Motivation: XML is widely used in business applications and is the de facto standard for information exchange among different enterprise information systems. Whenever the amount of processed XML data is a bottleneck, it is desirable that applications can directly query and update compressed XML data without having to decompress the data before accessing it.

There have been different contributions to the field of XML compressors generating queryable XML representations, that range from encoding-based [1] to schema-based [2], [3] to DAG-based [4] to grammar-based [5] compressed representations. We follow the grammar-based XML compression techniques, and we discuss how an XML compression technique, called CluX, can be extended by updates. Like the big majority of the XML compression techniques (e.g.[1],[2],[3],[5],[6],[7],[8],[9],[10],[11]), we assume that textual content of text nodes and of attribute nodes is compressed and stored separately and focus here on the compression of the structural part of an XML document.

Contributions: This paper proposes an approach to perform updates on grammar-compressed XML data directly, i.e., without prior decompression of the compressed data. Furthermore, our approach allows to perform several updates in parallel in such a way that e.g. performing 100 updates in parallel is more than 70 times faster than performing 100 updates sequentially.

We have implemented and evaluated updates on the compressed data. Our results show that it is not only possible to perform parallel updates on the CluX compressed data directly, but furthermore that in many cases, these updates can be performed in less time than it would take to decompress the compressed data, load the XML document, and compress the data again.

For simplicity of this presentation, we restrict it to XML documents containing only element nodes, i.e., attributes are regarded as special element types. Note

however that our implementation can handle full XML documents including attributes, text values, etc..

Paper Organization: The remainder of this paper is organized as follows. Section 2 describes the basic concept of grammar-based compression, i.e., how an XML tree can be stored in a more space saving way by sharing similar structures, it explains how these shared structures can be represented by patterns being used in tree grammars, and it describes how paths in XML document trees correspond to paths in tree grammars. Section 3 describes how updates can be performed on the compressed data directly and discusses the phases of performing updates: combining the update paths to an update DAG, isolating the update DAG from the grammar, updating isolated nodes, and sharing of identical sub-trees. Section 4 describes the evaluation of the presented update method. Section 5 compares our contribution to related work. Finally, Section 6 summarizes our contribution.

2 Sharing Similar Trees

2.1 The Paper’s Example Document

To simplify the following presentation, we do not distinguish between an XML node and its label. The following example is used for explaining the idea of grammar-based compression and to give a visual representation of our idea of direct updates on the compressed data.

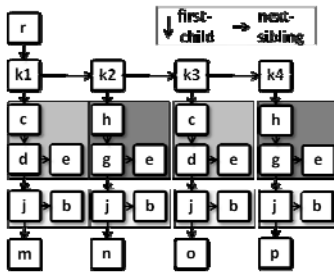


Fig. 1. Document tree of an XML document D with repeated matches of patterns

Fig. 1 shows an example XML document D represented as a binary tree, where e.g. r’s first-child is k1 whose next-sibling is k2. This XML document tree can be generated by the following grammar using the non-terminal S as the start symbol and the symbol ε as the empty sub-tree, i.e., the right hand side of the grammar rule is a term representing the pre-order notation of the binary tree given in Fig. 1:

$$S \rightarrow r(k1(c(d(j(m(\epsilon,\epsilon),b(\epsilon,\epsilon)),e(\epsilon,\epsilon)),\epsilon), k2(h(g(j(n(\epsilon,\epsilon),b(\epsilon,\epsilon)),e(\epsilon,\epsilon)),\epsilon), k3(c(d(j(o(\epsilon,\epsilon),b(\epsilon,\epsilon)),e(\epsilon,\epsilon)),\epsilon), k4(h(g(j(p(\epsilon,\epsilon),b(\epsilon,\epsilon)),e(\epsilon,\epsilon)),\epsilon,\epsilon))))))\epsilon)$$

Grammar 1: Grammar corresponding to the binary tree of Fig. 1

2.2 The Idea Behind Sharing Similar Trees

The simplest grammar-based XML compressors are those compressors that share identical sub-tree structures, such that the compressed grammar represents the minimal DAG of the XML tree [4].

Approaches like binary DAG compression, that share identical sub-trees T in an XML document D replace repeated occurrences of T in D by replacing each occurrence of T in D with a non-terminal N and adding a grammar rule that defines N to be a non-terminal that represents T .

In Grammar 1, there are four matches for each of the two patterns $b(\epsilon, \epsilon)$ and $e(\epsilon, \epsilon)$. Therefore, these matches can be replaced by the non-terminals B and E respectively, such that we get the following grammar:

$$\begin{aligned} S &\rightarrow r(k1(c(d(j(m(\epsilon, \epsilon), B), E), \epsilon), k2(h(g(j(n(\epsilon, \epsilon), B), E), \epsilon), \\ &\quad k3(c(d(j(o(\epsilon, \epsilon), B), E), \epsilon), k4(h(g(j(p(\epsilon, \epsilon), B), E), \epsilon), \epsilon))), \epsilon) \\ B &\rightarrow b(\epsilon, \epsilon) \\ E &\rightarrow e(\epsilon, \epsilon) \end{aligned}$$

Grammar 2: Grammar corresponding to the binary DAG of the XML tree of Fig. 1

Our approach goes beyond the idea of DAG compression and uses a parameterized grammar for sharing not only identical sub-trees, but even similar sub-trees. It follows the idea of grammar-based compression as it was introduced in BPLEX [5].

When looking for similar sub-trees having small differences, we find the three different patterns shown in Fig. 2(b) in the document tree D of Fig. 1: one pattern consisting of the nodes c , d , and e , another pattern consisting of the nodes h , g , and e , and a third pattern consisting of the nodes j and b respectively. For each of these patterns, there exist several matches in D which are highlighted in Fig. 2(a). Although the matches of the patterns have identical inner nodes, they cannot be shared in a DAG because the leaf nodes with labels m , n , o , or p respectively differ from each other.

Fig. 2 (b) shows the patterns $JB(X)$, $CDE(X)$ and $HGE(X)$, which consist of the nodes (j and b), (c , d , and e), and (h , g , and e) respectively. The nodes j , d , and g have a parameter ‘ X ’ as first-child.

The compression achieved by replacing the repeated patterns with a non-terminal can be seen when comparing Grammar 2 with Grammar 3 shown below. We express the pattern $JB(X)$ of Fig. 2(b) by one grammar rule with the left hand side $JB(X)$, where the parameter ‘ X ’ is being used for referencing the different child nodes m , n , o , and p of the j -nodes. This grammar rule is being used, e.g. when the term $j(m(\epsilon, \epsilon), B)$ occurring in the rule S of Grammar 2 is replaced with the term $JB(m(\epsilon, \epsilon))$ occurring in the rule S of Grammar 3. Here, $j(m(\epsilon, \epsilon), B)$ is called a *match*, and $JB(m(\epsilon, \epsilon))$ is called a *corresponding instantiation* of the pattern $JB(X)$.

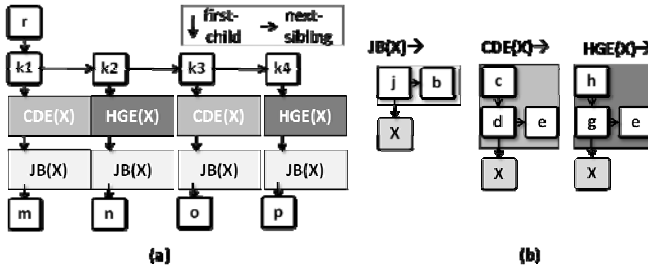


Fig. 2. (a) Example document of Fig. 1 with repeated patterns replaced by non-terminals. (b) Repeated patterns

By replacing each match with a corresponding instantiation, we get the following grammar, Grammar 3, which is more compact than Grammar 2:

$$\begin{aligned}
 S &\rightarrow r(k1(CDE(m(\epsilon,\epsilon)), k2(HGE(n(\epsilon,\epsilon)), k3(CDE(o(\epsilon,\epsilon)), k4(HGE(p(\epsilon,\epsilon)),\epsilon))),\epsilon) \\
 CDE(X) &\rightarrow c(d(JB(X),e(\epsilon,\epsilon)),\epsilon) \\
 HGE(X) &\rightarrow h(g(JB(X),e(\epsilon,\epsilon)),\epsilon) \\
 JB(X) &\rightarrow j(X,b(\epsilon,\epsilon))
 \end{aligned}$$

Grammar 3: A grammar sharing patterns by using parameterized rules

All terminal nodes except ϵ have two parameters, i.e. the first-child and the next-sibling. However, non-terminal nodes may have an arbitrary number of parameters.

2.3 Node Selection by Grammar Paths

The grammar path (GP): Each path to a selected node in any given XML document D corresponds to exactly one grammar path (GP) in the compressed grammar G of D . Intuitively, GP describes not only which grammar rules are called to find the selected node, but also from where in a given grammar rule, the next grammar rule is called. For this purpose, GP contains an alternating sequence of grammar rule names and index positions within these grammar rules of the occurrences of non-terminals N_i calling the next grammar rule. Additionally, the last number in GP is the index of the terminal symbol corresponding to the selected node in the last grammar rule collected by GP.

For example, if we apply the query $/k2//b$ to Grammar 3, GP is initially $[S]$, i.e. contains only the start rule. When $k2$ is found in the start rule S , no other grammar rule has been used, thus GP is still $[S]$. When the search for a descendant b of $k2$ continues via $k2$'s first-child, the 2nd non-terminal in the rule for S , i.e. $HGE(X)$, is called, and GP now is $[S,2,HGE(X)]$. Later, to find the first-child of g within the grammar rule for $HGE(X)$, the 1st non-terminal, i.e. $JB(X)$, is called. Finally, within the grammar rule for $JB(X)$, we pick the 2nd terminal symbol, i.e. the symbol b , to complete GP. This grammar path (GP), i.e. $[S,2,HGE(X),1,JB(X) : 2]$, corresponds to the b -node selected by the query $/k2//b$.

A formal definition of grammar paths however omitting the rule names is given in [12].

3 Parallel Updates on the Compressed Data

3.1 Basic Update Concepts and Parallel Update Problem Definition

The grammar DAG (GD): The grammar DAG (GD) visualizes from which non-terminal position within which grammar rule other grammar rules are called. In the grammar DAG (GD), there is one node N_i for each grammar rule G_i , and there is one edge (N_1, I, N_2) from node N_1 to node N_2 for each occurrence of a call of the grammar rule G_2 within the grammar rule G_1 , such that (counted from left to right) the I^{th} outgoing edge of N_1 refers to N_2 , if G_2 is the I^{th} non-terminal occurring in the right-hand side of G_1 . For example, the GD of Grammar 3 is shown in Figure 3.

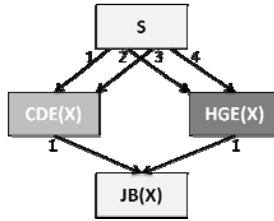


Fig. 3. Grammar DAG (GD) of Grammar 3

Each prefix P of a grammar path $GP=[P:t]$ through the given grammar G corresponds to one path in the GD. Based on this observation, we use GD for parallel updates on G .

For any given current context node ccn of the XML document, we simulate the following elementary update operations on the compressed grammar G , as more complex update operations can be constructed from these elementary update operations: delete ccn (del), re-label ccn with the new label y ($reLTo(y)$), insert $SubTree$ as first-child of ccn ($newFC(SubTree)$), insert $SubTree$ as next-sibling of ccn ($newNS(SubTree)$).

The Update Path (UP): For each elementary update operation $U \in \{ reLTo(z), delete, newFC(SubTree), newNS(SubTree) \}$ that is applied to a single selected XML document node being represented by a grammar path $GP=[N_1, I_1, \dots, N_{k-1}, I_{k-1}, N_k : t]$, we define a corresponding update path $UP = [(N_1', I_1, N_2'), \dots, (N_{k-1}', I_{k-1}, N_k'), (N_k', t, U)]$. The update path contains a new copy N_i' of each grammar path node N_i . Furthermore, we represent edges in the update path as triples $(Start, Index, End)$. Finally, we add an edge (N_k', t, U) from the copy N_k' of the last non-terminal N_k with the index position t of the terminal symbol in the grammar rule represented by N_k' to which the update operation U shall be applied. As a result, the final node of the update path contains the update operation U applied to the t^{th} terminal of N_k' .

For example, let the update operations be re-labeling all the nodes selected by $//h/b$ to z . Then, we get the update paths $[(S', 2, HGE(X)'), (HGE(X)', 1, JB(X)'), (JB(X)', 2, reLTo(z))]$ and $[(S'', 4, HGE(X)''), (HGE(X)'', 1, JB(X)''), (JB(X)'', 2, reLTo(z))]$.

The Parallel Update Problem Definition: Given a set of update operations, the parallel update problem is to compute a DAG of update paths and to isolate the DAG of update paths from the grammar G in parallel in order to keep the compressed grammar small and in order to keep the update process fast.

3.2 First Step of the Parallel Update Process: Constructing an Update DAG (UD)

Updating the grammar-compressed data is done by a two-step approach on the given grammar G .

In a first step, we navigate through G and identify the paths that have to be updated and combine them to an update DAG. Instead of collecting all the update paths individually while navigating through G , we combine all the update paths having a common prefix to construct an update tree.

To combine a collection of update paths into an update tree, first, all copies of the start node, i.e. S' and S'' in the previous example, are renamed to a single copy S' of the start node. This reflects the fact that all update paths that are combined into an update tree start at the same root node S' .

Thereafter, each set of common prefixes of multiple update paths is combined to a common prefix in the update tree as follows. Whenever the update tree contains two edges (N_j, I, N_j) and (N_i, I, N_k) where neither N_j nor N_k contains an update operation, the node N_k is renamed to N_j . This reflects the fact that both edges represent the unique I^{th} occurrence of a non-terminal in the grammar rule represented by N_j .

Continuing the previous example, the update tree has a common start node S' , but has no common edge of its two update paths.

In addition to combining updates with common prefixes within the update tree, we transform the update tree into an update DAG (UD). The UD is constructed bottom up from the update tree by sharing equal sub-trees. Two leaf nodes within the update tree are equal, if they have the same label, i.e. they contain the same update operation. Two inner nodes U_1 and U_2 of the update DAG are equal if they are copies of the same grammar rule and have similar outgoing edges, where two outgoing edges (U_1, I_1, U_3) and (U_2, I_2, U_4) from nodes U_1 and U_2 respectively are similar if $I_1=I_2$ and $U_3=U_4$.

Continuing the previous example, the sub-DAG rooted in $HGE(X)'$ is equal to the sub-DAG rooted in $HGE(X)''$. As similar edges are stored only once in the update DAG (UD), UD contains the edges $\{ (S', 2, HGE(X)'), (S', 4, HGE(X)'), (HGE(X)', 1, JB(X)'), (JB(X)', 2, reLTo(z)) \}$ and is shown in Figure 4.

3.3 Second Step of the Parallel Update Process: Isolating UD from GD

The second step of the parallel update process is to isolate the update DAG (UD) from the grammar DAG (GD) by isolating all the update paths contained in UD from GD in parallel. UD isolation is done by combining UD and GD into single DAG, called the extended update DAG (EUD), which is done by adding edges from certain UD nodes to certain GD nodes, such that after the extension, the EUD represents the result of isolating the original UD from GD. The UD isolation and update execution procedure is summarized in Algorithm 1 and is described in the following.

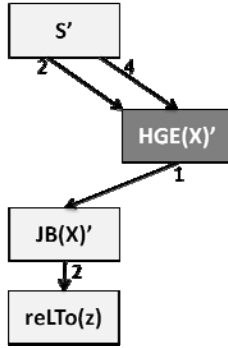


Fig. 4. Update DAG for re-labeling all the nodes of the document shown in Fig.1 that are selected by the query //h//b to z

- (1) for each non-leaf node U_i in UD do
- (2) { $N_i = \text{corresponding_GD_node}(U_i)$;
- (3) for each edge (N_i, I, N_k) in GD
- (4) if (no edge (U_i, I, U_k) to any node U_k exists in UD) add (U_i, I, N_k) to UD
- (5) }
- (6) EUD = UD ;
- (7) Perform updates on EUD as described in Section 3.4.
- (8) Share identical nodes on EUD as described in Section 3.5.
- (9) Top-down for each node N_i in GD do
- (10) if $(N_i \text{ has no incoming edges })$ delete N_i and all outgoing edges of N_i
- (11) return EUD ;

Algorithm 1. Update DAG (UD) isolation from a grammar DAG (GD)

For each non-leaf node U_i in the update DAG, let N_i be the corresponding node in the grammar DAG, i.e., U_i and N_i belong to the same grammar rule G_i (line (2)). Note that different nodes of the update DAG may correspond to the same node N_i . For each outgoing edge (N_i, I, N_k) of N_i for which no edge (U_i, I, U_k) exists in the update DAG, add an edge (U_i, I, N_k) to the update DAG (lines (3)-(4)). That is, an outgoing edge from U_i to the GD node N_k representing G_i 's I^{th} non-terminal is added to each UD node U_i for which an edge to a UD node U_k representing G_i 's I^{th} non-terminal does not yet exist, such that finally the UD represents the same number of grammar paths as the GD. On the UD with this extensions, called EUD (line (6)), the update and sharing operations described in the following sections are performed. After computing the EUD and performing all the update and sharing operations, some GD nodes and GD edges are reachable by a path from the UD root and others are not. As the non-reachable GD nodes and GD edges are useless, they are deleted from GD (lines (9)-(10)). The remaining and returned extended update DAG (line (11)) represents the result of isolating the original UD from the GD and performing the updates on the resulting EUD.

Fig. 5 shows the continued example of the isolation process for re-labeling the nodes selected by the XPath expression //h//b to z. The input of the isolation process is the grammar DAG (as shown in Fig. 3) and the update DAG (UD) shown in Fig. 4 and containing the following set of edges: $\{ (S',2,HGE(X)'), (S',4,HGE(X)'), (HGE(X)', 1, JB(X)'), (JB(X)',1,relTo(z)) \}$.

For the root node S' of the UD, the corresponding node S of the GD has two additional outgoing edges at the index positions 1 and 3 to the node $CDE(X)$. Therefore, within lines (2)-(4), Algorithm 1 extends the UD with the edges $(S',1,CDE(X))$ and $(S',3,CDE(X))$ from S' to $CDE(X)$.

For the UD node $HGE(X)'$, the corresponding node $HGE(X)$ in GD has no additional outgoing edge, i.e. no outgoing edge from $HGE(X)'$ has to be added. That also applies to the UD node $JB(X)'$.

Now, the isolation phase is completed, and replacing the terminal symbol b with z within the copied grammar rule represented by the node $JB(X)'$ modifies exactly the set of paths selected by the XPath query //h//b to have the new label z , and removes the node $relTo(z)$ from the EUD.

Finally, the nodes S and $HGE(X)$ and their outgoing edges can be deleted from GD as they are not reachable from any path starting in S' . The resulting UD is shown in Fig. 5(b).

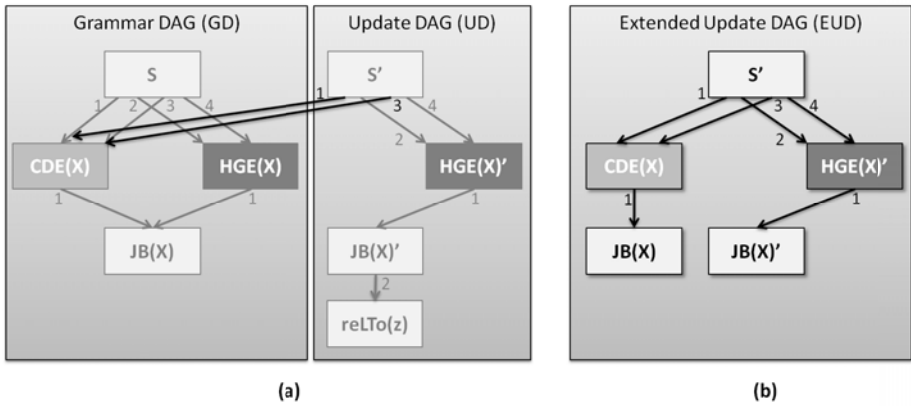


Fig. 5. Isolation of nodes selected by //h//b for re-labeling them to z

Note that we implement UD nodes by copying grammar rules from the grammar represented by GD. For example, Grammar 3 is modified in such a way that the rules S' , $HGE(X)'$, and $JB(X)'$ are copied from the rules S , HGE , and $JB(X)$ respectively, and $HGE(X)'$ instead of $HGE(X)$ is called from the S' rule, and $JB(X)'$ instead of $JB(X)$ is called from the $HGE(X)'$ rule. Then, b can be replaced with z within the $JB'(X)$ rule in the following update step.

3.4 Performing the Updates

After UD isolation by extending the UD to a EUD in lines (1)-(6) of Algorithm 1, each edge (U_i, I, U) in EUD to an update leaf node U , i.e. a leaf node of EUD where

U is an update operation, is used for modifying the grammar rule G_i represented by U_i . The update operation U is applied to the I^{th} terminal symbol T of G_i , e.g., for the edge $(\text{JB}(X)', 2, \text{reLTo}(z))$, the grammar rule $\text{JB}(X)'$ is updated by replacing the 2^{nd} terminal symbol, i.e. b, with z. Depending on the particular update operation U described by the edge (U_i, I, U) , we do the following.

If $U = \text{reLTo}(z)$, we substitute the I^{th} terminal symbol of G_i , i.e. T, with z.

If $U = \text{newFC}(\text{cst})$ or $U = \text{newNS}(\text{cst})$, i.e., if the update requires inserting a compressed sub-tree cst as a first-child or as a next-sibling of T respectively, we have to set the current first-child or next-sibling of T as the new next-sibling of the root of cst. As we might insert the same sub-tree cst at several places within the original XML document, the most efficient way to do this is to set the next-sibling of the root rule of cst (which is a null pointer, i.e. ϵ , before the insertion) to a parameter, and to replace the first-child fc or the next-sibling ns of T by a call of the root rule of cst with the parameter value fc or ns respectively.

If we consider for example the node 'k1' as selected node in Grammar 3 and want to insert a node $z(\epsilon, \epsilon)$ as the first-child of /k1, i.e., cst consists of the single rule $Z(X) \rightarrow z(\epsilon, X)$, we would insert a new rule $Z(X) \rightarrow z(\epsilon, X)$ into Grammar 3 and replace the current first-child cfc of k1 in Grammar 3 with a call $Z(\text{cfc})$ of this new rule. Thereby, cfc becomes the next-sibling of z. This means, that the call 'k1(CDE(m(ϵ, ϵ)),...)' within the start rule S of Grammar 3 is replaced with the call 'k1(Z(CDE(m(ϵ, ϵ)),...))'.

If $U = \text{del}$, i.e., the update requires deleting the sub-tree having its root in T, we can simply replace $T(\text{FC}, \text{NS})$ with T's own next-sibling NS in the grammar rule represented by U_i . If we remove a formal parameter from a rule during the deletion, we have to delete the corresponding actual parameter within each call of the modified rule as well. In the worst case, i.e., if the actual parameter of a rule is defined in the rule represented by S' , we have to modify all the rules represented by nodes of EUD that lay on paths from S' to U_i within the EUD.

For example, if we delete the nodes of the sub-trees, the root of which is selected by //d//j, i.e. is the j node with first child m or the j node with first child o, UD contains the edges $(S', 1, \text{CDE}(X)')$, $(S', 3, \text{CDE}(X)')$, $(\text{CDE}(X)', 1, \text{JB}(X)')$, and $(\text{JB}(X)', 1, \text{del})$. By applying the delete operation to the first non-terminal of the $\text{JB}(X)'$ rule, i.e. to j, the right-hand side of this rule is replaced with $b(\epsilon, \epsilon)$. As now the $\text{JB}(X)'$ rule does not contain a parameter anymore, the parameter has to be removed from the $\text{CDE}(X)'$ rule calling it too. And finally, we have to delete the parameters used in the rule calls of rule $\text{CDE}(X)'$ within the S' rule too. By doing this, the first-child nodes of nodes //d//j with labels m and o are deleted as well.

3.5 Sharing Identical Nodes

Although the initial grammar to be updated does not contain anymore sub-structures that can be shared, during the update process new sub-structures are generated that might be similar or identical to already existing sub-structures.

For example, if we re-label in our example the node /k2//g to 'c' and the node /k2//h to 'd', after the UD isolation and the update process, the grammar would contain a rule

$$\text{HGE}'(X) \rightarrow c(d(\text{JB}(X), e(\epsilon, \epsilon)), \epsilon)$$

which is identical to the rule $CDE(X)$. Therefore, we can delete the rule $HGE'(X)$ and replace each call of it by a call of the rule $CDE(X)$. The grammar using rule $HGE'(X)$ is correct and can be decompressed and processed correctly, but after replacing $HGE'(X)$ with $CDE(X)$, the grammar is more compact, i.e., the compression ratio is optimized. For this purpose, we perform a sharing phase after the updating phase.

In order to find redundant rules, we could compare the modified rule with all existing and modified rules. But this comparison becomes more efficient, when we use the information given by the EUD. Two rules can only be identical, if they call the same sequence of other grammar rules. For the EUD, this means that we only have to compare these rules that have the same sequence of children within the EUD. This reduces the number of comparisons within the sharing phase.

4 Evaluation

All tests were performed on an Intel Core2 Duo CPU P870 @ 2,53 GHz with 4 GB of RAM running our prototype on Java 1.6.

In a first series of measurements, we compared the compression strength of CluX with two other approaches, *gzip* and *bzip2*, based on the following XML datasets:

1998statistics (1998 – 656 kB) – Baseball statistics of the year 1998, catalog-01 (C1 – 10.4 MB) and dictionary-01 (D1 – 10.4 MB) – documents generated by the Xbench benchmark, hamlet (H – 273 kB) – the Shakespeare play, JST_snp.chr (JST – 35.5 MB) – data on the tumor suppressor gene JST, and NCBI_gene.chr (NCBI – 23.0 MB) – data from the National Center for Biotechnical Information, Treebank (TB – 51.9 MB) – a parsed text corpus, and XMark (XM – 111.1 MB) – a document that models auctions.

Usually, CluX compresses best (c.f. Fig. 6), followed by *bzip2*, and finally followed by *gzip*.

In a second series of measurements, we have compared the time for direct updates on the compressed data to the sum of the times needed for decompression, loading the uncompressed document as a DOM tree into main memory (i.e., no updates were performed) and recompression when using CluX, *bzip2* or *gzip* as compression tool.

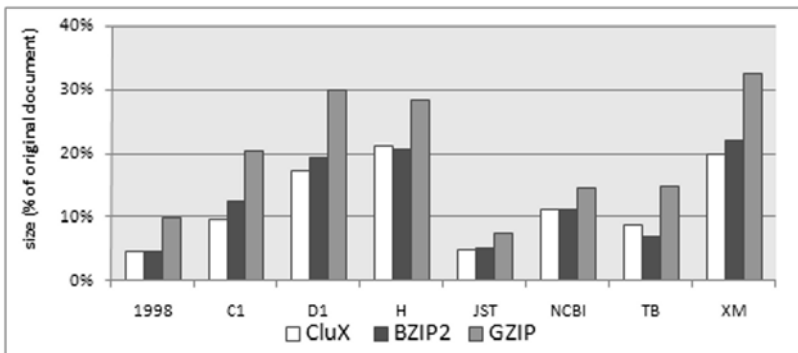


Fig. 6. Compression ratios of CluX compared with *bzip2* and *gzip*

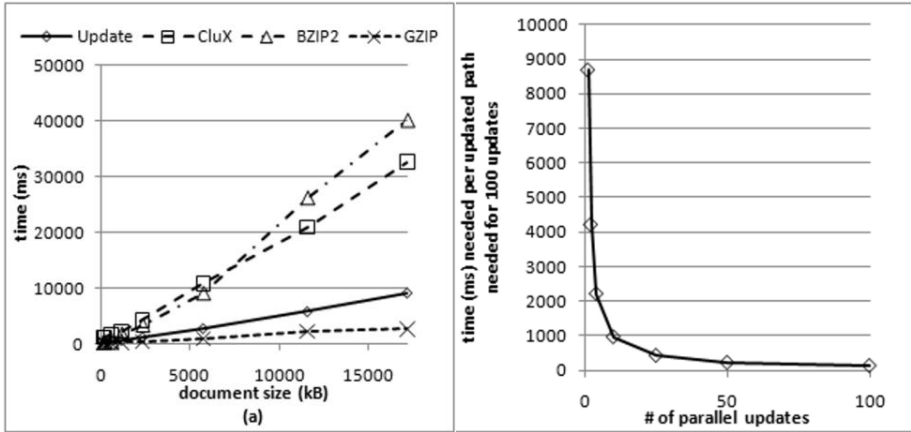


Fig. 7. (a) Update time of CluX compared to compression and decompression times of CluX, bzip2 and gzip, (b) Update time required for a scaling number of parallel updates

With scaling the document size (c.f. Fig. 7(a)), the direct updates on CluX can be performed faster than the compression and decompression of CluX and bzip2. For a document with a size of 15 MB, the update on the compressed data is 3.5 times faster than the decompression and recompression by CluX and 4.4 times faster than the decompression and recompression by bzip2. Only gzip, that reaches a far weaker compression ratio than CluX can be decompressed and recompressed in less time than the update process directly on the compressed data requires. Finally, we have examined the impact of parallel updates compared to sequential updates. For this purpose, we randomly selected 100 paths of the grammar DAG and relabeled the XML node defined by these paths. Fig. 7(b) shows that performing 100 updates in parallel as a multi-update operation is more than 70 times faster than performing 100 updates sequentially.

5 Related Work

Besides generic compressors like gzip, bzip2 or 7zip (based on LZMA) all of which do not allow direct query evaluation on the compressed data, there are several approaches to XML structure compression. XML structure compression can be mainly divided into three categories: encoding-based compressors, schema-based compressors and grammar-based compressors.

The encoding-based compressors allow for a faster compression speed than the other ones, as only local data has to be considered in the compression as opposed to considering different sub-trees as in grammar-based compressors. Examples for encoding-based approaches are the approaches [13], [6], and [7], XMill [8], XPRESS [9], XGrind [14], and [1]. Whereas XMill is not queryable, i.e., it does not support the navigation or the evaluation of XPath queries on the compressed document directly, i.e., without prior decompression, all other approaches are queryable.

Schema-based compression comprises such approaches as XCQ [2], XAUST [15], Xenia [3], and XSDS [10]. They subtract the given schema information from the structural information. Instead of a complete XML structure stream or tree, they only

generate and output information not already contained in the schema information (e.g., the chosen alternative for a choice-operator or the number of repetitions for a *-operator within the DTD). These approaches are queryable and applicable to XML streams, but they can only be used if schema information is available.

XQzip [11] and the approaches presented in [16] and [4] belong to grammar-based compression. They compress the data structure of an XML document by combining identical sub-trees.

An extension of [4] and [11] is the BPLEX algorithm [5]. This approach not only combines identical sub-trees, but recognizes similar patterns within the XML tree, and therefore allows a higher degree of compression. The approach presented in this paper, which is an extension of [17], follows the same idea. But instead of combining similar structures bottom-up, our approach searches within a given window the most promising pair to be combined while following one of three possible clustering strategies. Furthermore, in contrast to [12] and [18], that performs updates by path isolation only sequentially, our approach allows performing updates in parallel which takes only a fraction of time.

6 Summary and Conclusions

We have shown how updates can be performed directly on CluX, a clustering-based compression approach for XML trees, i.e., without the need to decompress the compressed data in advance. As an XML file compressor, CluX compresses on average 70% better than the generic compressor gzip and 5% better than the generic compressor bzip2. CluX compression can be applied to infinite data streams – and in contrast to gzip or bzip2, path queries and updates can be evaluated directly on the compressed representation, i.e., without prior decompression. Beyond other clustering or multiplexing based approaches like e.g. the BPLEX algorithm [12], [5], CluX offers an update DAG isolation technique that allows to perform several updates in parallel, and our evaluation has shown that performing 100 updates in parallel takes significantly less time than performing 100 updates sequentially. Furthermore, our evaluation on a file with a size of 15 MB has shown that performing the updates directly on the compressed data with our update algorithm is more than 3 times faster than decompressing the data first and recompressing it with CluX, and it is more than 4 times faster than the decompression and recompression with bzip2.

We furthermore believe that this technique of performing several updates in parallel on the compressed data directly is not restricted to CluX, but can be extended to DAG-based compressors like [5] and to other grammar-based compressors like e.g. BPLEX [5], the main idea of which is to share similar sub-trees.

References

1. Zhang, N., Kacholia, V., Özsu, M.: A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML. In: Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, Boston, MA, USA, pp. 54–65 (2004)
2. Ng, W., Lam, W., Wood, P., Levene, M.: XCQ: A queryable XML compression system. *Knowl. Inf. Syst.*, 421–452 (2006)

3. Werner, C., Buschmann, C., Brandt, Y., Fischer, S.: Compressing SOAP Messages by using Pushdown Automata. In: 2006 IEEE International Conference on Web Services (ICWS 2006), Chicago, Illinois, USA, pp.19–28 (2006)
4. Buneman, P., Grohe, M., Koch, C.: Path Queries on Compressed XML. In: Proceedings of 29th International Conference on Very Large Data Bases, Berlin, Germany, pp. 141–152 (2003)
5. Busatto, G., Lohrey, M., Maneth, S.: Efficient Memory Representation of XML Documents. In: Bierman, G., Koch, C. (eds.) DBPL 2005. LNCS, vol. 3774, pp. 199–216. Springer, Heidelberg (2005)
6. Cheney, J.: Compressing XML with Multiplexed Hierarchical PPM Models. In: Proceedings of the IEEE Data Compression Conference (DCC 2001), Snowbird, Utah, USA, p. 163 (2001)
7. Girardot, M., Sundaresan, N.: Millau: an encoding format for efficient representation and exchange of XML over the Web. *Computer Networks* 33, 747–765 (2000)
8. Liefke, H., Suciu, D.: XMILL: An Efficient Compressor for XML Data. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, Texas, USA, pp. 153–164 (2000)
9. Min, J.-K., Park, M.-J., Chung, C.-W.: XPRESS: A Queriable Compression for XML Data. In: Halevy, A., Ives, Z., Doan, A. (eds.) Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, pp. 122–133 (2003)
10. Böttcher, S., Hartel, R., Messinger, C.: XML Stream Data Reduction by Shared KST Signatures. In: 42st Hawaii International International Conference on Systems Science (HICSS-42 2009), Proceedings (CD-ROM and online), Waikoloa, Big Island, HI, USA, pp. 1–10 (2009)
11. Cheng, J., Ng, W.: XQzip: Querying Compressed XML Using Structural Indexing. In: Hwang, J., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 219–236. Springer, Heidelberg (2004)
12. Fisher, D., Maneth, S.: Structural Selectivity Estimation for XML Documents. In: Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, Istanbul, Turkey, pp. 626–635 (2007)
13. Bayardo Jr., R., Gruhl, D., Josifovski, V., Myllymaki, J.: An evaluation of binary XML encoding optimizations for fast stream based xml processing. In: Feldman, S., Uretsky, M., Najork, M., Wills, C. (eds.) Proceedings of the 13th International Conference on World Wide Web, New York, NY, USA, pp. 345–354 (2004)
14. Tolani, P., Haritsa, J.: XGRIND: A Query-Friendly XML Compressor. In: Proceedings of the 18th International Conference on Data, ICDE, San Jose, CA, pp. 225–234 (2002)
15. Subramanian, H., Shankar, P.: Compressing XML Documents Using Recursive Finite State Automata. In: Farré, J., Litovsky, I., Schmitz, S. (eds.) CIAA 2005. LNCS, vol. 3845, pp. 282–293. Springer, Heidelberg (2006)
16. Adiego, J., Navarro, G., Fuente, P.: Lempel-Ziv Compression of Structured Text. In: Data Compression Conference, Snowbird, UT, USA, pp. 112–121 (2004)
17. Böttcher, S., Hartel, R., Krislin, C.: CluX - Clustering XML Sub-trees. In: ICEIS 2010 - Proceedings of the 12th International Conference on Enterprise Information Systems, Funchal, Madeira, Portugal, pp. 142–150 (2010)
18. Damien, F., Maneth, S.: Selectivity Estimation. Patent WO 2007/134407 A1 (May 2007)

Reverting the Effects of XQuery Update Expressions

Federico Cavalieri¹, Giovanna Guerrini¹, and Marco Mesiti²

¹ DISI – University of Genova
{cavalieri, guerrini}@disi.unige.it

² DICO – University of Milano
mesiti@dico.unimi.it

Abstract. The need of reverting the effects of updates on the affected documents arises in many contexts, ranging from undos in transactional applications to versioning systems. In this paper, we investigate this issue for XQuery Update expressions, relying on the Pending Update List (PUL) obtained from the evaluation of an expression on a document. Specifically, we introduce an inversion operator, that, given a PUL to be applied on a document, allows to determine a corresponding inverted PUL that, applied on the modified document, produces the original document. Moreover, an alternative approach for enriching a PUL with additional information, so that it can be inversely applied, is proposed and the two approaches are experimentally compared.

Keywords: XML, Updates, Dynamic reasoning, Update processing.

1 Introduction

The ability of reverting the effects of an update operation is useful in many situations. Consider for instance a distributed transactional context, in which transactions can be aborted and rolled back, and thus the corresponding operations need to be undone. More flexible update processing approaches based on check-in/check-out policies, such as those employed in collaborative document editing, may benefit as well of such ability. It may also become crucial in versioning contexts, if versions are handled by recording the updates that transformed a version into the following one (edit-based approaches according to [4]) instead of the various data snapshots. In this context, update reversion is the basis for moving across different versions.

In this paper, we investigate this problem in the context of updates on XML documents expressed as XQuery Update (XQU) expressions [15]. The evaluation of an XQU expression on a document produces a set of atomic update requests, represented as a Pending Update List (PUL), that is then applied on the document. In [3] we discussed the relevance of contexts (such as collaborative editing, disconnected execution, data clouds) in which updates are not necessarily executed right after and on the same server where the update expression requesting them is evaluated. Thus, the process of expressing and requesting updates is decoupled from that of making them effective on documents. PULs can be produced by a machine, sent over a network, saved to disks, and later applied on the document, possibly by a different machine. Referring to update reverting, this means that the effects of an update expression can be discarded on a server different from the one on which they have been applied.

According to this processing model, there are two alternative approaches to revert the effects of an update expression. The first one is to invert a PUL through a PUL inversion operator. This operator, given a PUL Δ on a document D , produces another PUL Δ^{-1} , that, when executed on the document updated according to Δ , returns the original document D , thus undoing the updates in Δ . This approach however requires to access document D for producing the inverse. An important feature of the PUL operators in [3], by contrast, is document independence: operators on PULs should not require, whenever possible, to access the document. They rely on structural information on the document that is incorporated in the PUL itself through a labeling scheme. Thus, an alternative approach, to be exploited in contexts where the need of reverting update effects is known to be likely to arise, is to modify the evaluation of XQU expressions so that they produce *completed PULs* rather than PULs. A completed PUL Δ^{\leftrightarrow} contains the information for being applied either forward (to actually apply updates) or backward (to revert their effects), and in both cases it can be applied in streaming.

The paper investigates both approaches, proposing a set of inversion rules and a PUL inversion algorithm, defining completed PULs, and discussing their forward and backward streaming application. Both approaches have been implemented, by modifying the Qizx [14] library to produce PULs and completed PULs (both represented as XML documents) and to accept them as input.

The paper is organised as follows. Section 2 introduces some preliminary notions on XML documents and PULs. PUL inversion is discussed in Section 3, whereas Section 4 introduces completed PULs and their backward and forward application. Section 5 experimentally compares the two approaches. Section 6 contrasts our approach with related work. Some concluding remarks are finally presented.

2 Preliminaries

In this section we introduce the adopted representations of XML documents, define PULs of operations, their semantics, and discuss their streaming evaluation.

2.1 XML Document Representations

A document can be represented as a labeled tree. A document D is a tuple $(V, \gamma, \lambda, \nu)$ where: V is a set of nodes representing elements, attributes or text nodes (for simplicity, only these types among those in [15] are considered); γ is a function associating with each node its children; λ and ν are labeling functions associating with each element and attribute node a name in a set \mathcal{N} and with each text and attribute node a value in a set \mathcal{V} , respectively. Auxiliary functions V and \mathcal{R} denote the nodes and the root(s) of a single tree D or of a collection of trees, respectively, and τ assigns to each node v in V a value in the set $\{e, a, t\}$ denoting its type. Moreover, auxiliary functions LS , P and T assign to each node its adjacent left sibling, parent and subtree, respectively, when they exists (\perp , otherwise). Coherently with the XDM model, the attribute value is seen as a property of the attribute node, whereas textual contents of elements are modeled by separate nodes. A unique and immutable identifier is associated with each node in V , and, wherever no confusion arises, we do not distinguish nodes from their identifiers.

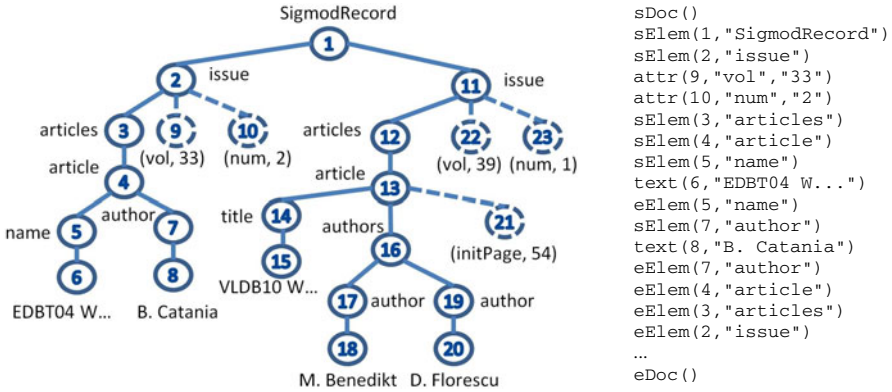


Fig. 1. XML Document representation: tree-based (left), event-based (right)

Alternatively, a document can be represented as a stream of SAX-like events, which describe the nodes encountered in a depth-first traversal of the document. A document D is an ordered sequence of events: $sDoc()$ and $eDoc()$, indicating the start/end of the document; $sElem(v,n)$ and $eElem(v,n)$, indicating the start/end of element v tagged n ; $attr(v,n,s)$ and $text(v,s)$, indicating the attribute/text node v with name n and value s . Fig. 1 contrasts the tree/event-based representations of a fragment of the `SigmodRecord` document to be updated. In the former, dotted lines are used to represent edges leading to attribute nodes.

In handling PULs, we need to check whether some relationships (like parent-child, element-attribute, left-right sibling) hold among two nodes in both representations. This information is obtained without directly accessing the document through a labeling scheme [12] associated with nodes through function l . In this scheme, the introduction/deletion of nodes does not require node re-labelling. Table 1 reports the predicates that can be assessed through the adopted labeling scheme.

2.2 Update Operations and PULs

Table 2 reports the primitives defined in [15] that we consider, where $v \in V$ is a node, $P = [T_1, \dots, T_k]$, $k \geq 0$, is a list (possibly empty in case of the `repN` or `repC` operation) of trees, $n \in \mathcal{N}$ is a name, $s \in \mathcal{V}$ is a value. Given an operation op , $t(op)$ denotes its target node, $o(op)$ denotes its name, and $p(op)$ denotes its second parameter

Table 1. Structural relationships

Predicate	Description
$v_1 \triangleleft v_2$	v_1 precedes v_2 in document order
$v_1 \triangleleft_s v_2$	v_1 is the (adjacent) left sibling of v_2
$v_1 \blacktriangleleft_s v_2$	v_1 is a preceding sibling of v_2
$v_1 /_c v_2$	v_1 is a child of v_2
$v_1 /_a v_2$	v_1 is an attribute of v_2
$v_1 //_d v_2$	v_1 is a descendant of v_2
$v_1 //_{\bar{d}} v_2$	v_1 is a descendant of v_2 but not an attribute of v_1

Table 2. Update operations

Operation	Description	Conditions	Completed Operation
$\text{ins}^{\leftarrow}(v, P)$ $\text{ins}^{\rightarrow}(v, P)$	Insert the trees in P before/after node v	$\tau(v) \neq \mathbf{a}, \forall r \in \mathcal{R}(P) \tau(r) \neq \mathbf{a}$	idem
$\text{ins}^{\swarrow}(v, P)$ $\text{ins}^{\searrow}(v, P)$	Insert the trees in P as first/last children of node v	$\tau(v) = \mathbf{e}, \forall r \in \mathcal{R}(P) \tau(r) \neq \mathbf{a}$	idem
$\text{ins}^{\downarrow}(v, P)$	Inserts the trees in P as children of node v , in an implementation defined position	$\tau(v) = \mathbf{e}, \forall r \in \mathcal{R}(P) \tau(r) \neq \mathbf{a}$	idem
$\text{insA}(v, P)$	Inserts the trees in P as attributes of node v	$\tau(v) = \mathbf{e}, \forall r \in \mathcal{R}(P) \tau(r) = \mathbf{a}$	idem
$\text{del}(v)$	Deletes node v		$\text{del}(v, T(v))$
$\text{repN}(v, P)$	Replaces node v with the trees in P (possibly none)	$\forall r \in \mathcal{R}(P) (\tau(r) = \tau(v) = \mathbf{a})$ $\vee (\tau(v) \neq \mathbf{a} \wedge \tau(r) \neq \mathbf{a})$	$\text{repN}(v, P, T(v))$
$\text{repV}(v, s)$	Replaces the value of node v with $s \in \mathcal{V}$	$\tau(v) \in \{\mathbf{t}, \mathbf{a}\}$	$\text{repV}(v, s, \nu(v))$
$\text{repC}(v, v')$	Replaces the children of node v with text node v' or with nothing	$\tau(v) = \mathbf{e} \wedge (v' = \perp \vee \tau(v') = \mathbf{t})$	$\text{repC}(v, v', T(\gamma(v)))$
$\text{ren}(v, n)$	Renames node v with $n \in \mathcal{N}$	$\tau(v) \in \{\mathbf{e}, \mathbf{a}\}$	$\text{ren}(v, n, \lambda(v))$

(undefined if $o(op) = \text{del}$). An operation op is applicable on a document D if its target belongs to D and the applicability conditions (identified in [3]) of op hold. The meaning of last column will be discussed in Section 4.

A *pending update list (PUL)* [15] is an unordered list of operations among those in Table 2. Since the order of operations is irrelevant, some pairs of operations cannot occur in the same PUL. Specifically, no pairs of replacement operations of the same type with the same target (referred to as *incompatible operations*) can occur. For a PUL to be applicable on a document (cf. function `applyUpdates` in [15]) it must contain no incompatible operations and all its operations must be applicable on the document.

Operation semantics is specified in [11]. The semantics of operation ins^{\downarrow} is non-deterministic since the actual position of the inserted nodes group in the target node is not univocally specified. Thus, the application of an operation op to a document D produces one document in a set of *obtainable documents*, denoted as $\mathcal{O}(op, D)$.

The semantics of a PUL Δ on a document D is obtained by applying the operations in Δ in five stages [11]. At each stage, a subset of the operations are applied to enforce the precedence relation on operation types specified in [15]. The order of application of operations within each stage is not prescribed by [15]. Thus, when multiple insertion operations of the same type with the same target appear in the same PUL, the relative order of the inserted groups of nodes is not fixed as well. Therefore, the cardinality of $\mathcal{O}(\Delta, D)$ is greater than one when ins^{\downarrow} occurs in Δ or Δ contains more than one insertion operation of the same type on the same target.

Example 1. Let D be the document in Fig. 1. Operation $op_1 = \text{del}(14)$ is deterministic and thus $\mathcal{O}(op_1, D)$ is a singleton. Operation $op_2 = \text{ins}^{\downarrow}(16, \langle \text{author} \rangle \text{G.Guerrini} \langle / \text{author} \rangle)$, by contrast, may lead to inserting the element as first, second, or last author of the second paper, thus $\mathcal{O}(op_2, D)$ contains three documents. Finally, $|\mathcal{O}(\Delta, D)| = 6$ for $\Delta = \{\text{ins}^{\downarrow}(16, \langle \text{author} \rangle \text{G.Guerrini} \langle / \text{author} \rangle), \text{ins}^{\searrow}(4, \langle \text{initP} \rangle 132 \langle / \text{initP} \rangle), \text{ins}^{\searrow}(4, \langle \text{lastP} \rangle 134 \langle / \text{lastP} \rangle)\}$.

2.3 PUL Streaming Application

Given a document represented as a sequence of events E , a PUL is modeled as an *event transformer*, which transforms E in a new sequence of events corresponding to one of the obtainable documents. An empty PUL corresponds to an identity transformation. Each operation in a non-empty PUL requires some event transformation. The order in which transformations must be applied on the events of a node v is the same as in the staged execution of an XQU expression.

Example 2. Consider the PUL $\Delta = \{\text{repC}(2, []), \text{ren}(2, \text{"issues"})\}$ and the document in Fig. 1. The first operation requires to remove any non-attribute event which occurs between $\text{sElem}(2, _)$ and $\text{eElem}(2, _)$. The second one requires to alter these two events replacing the name. The transformed event sequence is: $\text{sDoc}()$, $\text{sElem}(1, \text{"SigmodRecord"})$, $\text{sElem}(2, \text{"issues"})$, $\text{attr}(9, \text{volume}, \text{"33"})$, $\text{attr}(10, \text{number}, \text{"2"})$, $\text{eElem}(2, \text{"issues"})$, ..., $\text{eDoc}()$.

3 PUL Inverse

In this section we discuss the inversion of operations and PULs.

3.1 Operation Inversion

The inversion of an operation op applicable on a document D produces one or more operations that revert the modifications made by op on D . To revert the effects of a single operation, multiple operations may be required. Consider the case of a list of subtrees inserted through a single ins : each single subtree must be separately deleted.

Definition 1 (Operation Inverse). *Let op be an operation applicable on a document D . An inverse of op on D is a PUL Δ_{op}^{-1} s.t.: $\forall D' \in \mathcal{O}(op, D) : \mathcal{O}(\Delta_{op}^{-1}, D') = \{D\}$.*

Different PULs can be identified to revert the effects of each operation. The following approach has been devised: ins is reverted by removing all inserted subtrees, repV/ren by restoring the original value/name of the target node, repC by restoring the original node children. For node deletions and replacements the introduced nodes must be deleted, while removed nodes must be placed back in their original position. Attribute nodes are inserted by an insA , while other nodes through ins^- , if an adjacent left sibling exists in D , through an ins^{\setminus} otherwise. Table 3 defines inverse operations.

Example 3. Consider the document in Fig. 1 and the following operations (the node identifier is reported as superscript of the node). $op_1 = \text{ren}(5, \text{"title"})$, $op_2 = \text{del}(7)$, $op_3 = \text{ins}(4, \langle \text{author} \rangle X^{25} \langle / \text{author} \rangle^{24}, \langle \text{author} \rangle Y^{27} \langle / \text{author} \rangle^{26})$. The inverses are: $\Delta_{op_1}^{-1} = \{\text{ren}(5, \text{"name"})\}$, $\Delta_{op_2}^{-1} = \{\text{ins}^- (5, \langle \text{author} \rangle B.Catania^8 \langle / \text{author} \rangle^7)\}$, $\Delta_{op_3}^{-1} = \{\text{del}(24), \text{del}(26)\}$, respectively.

Proposition 1 (Correctness of Operation Inversion). *For any operation op applicable on a document D , let Δ_{op}^{-1} be the inverse PUL obtained according to Table 3. Then Δ_{op}^{-1} is an inverse of op on D , according to Definition 1.*

Proof Sketch. This can be straightforwardly proved by individually considering the semantics of each operation and of the corresponding inverse as defined in Table 3.

Table 3. Operation inverses

Operation	Inverse	Condition
$\text{ins}^r(v, [T_1, \dots, T_n])$	$\{\text{del}(\mathcal{R}(T_1)), \dots, \text{del}(\mathcal{R}(T_n))\}$	$r \in \{\leftarrow, \rightarrow, \downarrow, \swarrow, \searrow, \mathbf{A}\}$
$\text{repV}(v, s)$	$\{\text{repV}(v, v(v))\}$	
$\text{ren}(v, l)$	$\{\text{ren}(v, \lambda(v))\}$	
$\text{repC}(v, v')$	$\{\text{repN}(v', \gamma(v))\}$ $\{\text{ins}^\downarrow(v, \gamma(v))\}$	$v' \neq \square$ $v' = \square$
$\text{del}(v)/\text{repN}(v, \square)$	$\{\text{insA}(P(v), T(v))\}$ $\{\text{ins}^\rightarrow(LS(v), T(v))\}$ $\{\text{ins}^\swarrow(P(v), T(v))\}$	$\tau(v) = \mathbf{a}$ $\tau(v) \neq \mathbf{a} \wedge LS(v) \neq \perp$ $\tau(v) \neq \mathbf{a} \wedge LS(v) = \perp$
$\text{repN}(v, [T_1, \dots, T_n])$	$\{\text{repN}(\mathcal{R}(T_1), T(v)), \text{del}(\mathcal{R}(T_2)), \dots, \text{del}(\mathcal{R}(T_n))\}$	

3.2 PUL Inversion

Starting from the operation inverse, we define the PUL inverse.

Definition 2 (PUL Inverse). Let Δ be a PUL applicable on a document D . An inverse of Δ on D is a PUL Δ^{-1} s.t. $\forall D' \in \mathcal{O}(\Delta, D) : \mathcal{O}(\Delta^{-1}, D') = \{D\}$.

In inverting a PUL Δ , the following properties must be guaranteed: (i) each inverse operation must be applicable in all the obtainable documents; (ii) in case an operation in Δ is overridden (that is, it has no effect on the document), its inverse must have no effect; and, finally, (iii) the relative order of the nodes removed by Δ must be restored by its inverse. As shown by the following example, simply inverting each single operation in Δ independently does not allow to guarantee these properties.

Example 4. Consider the PUL $\Delta = \{\text{del}(5), \text{repV}(6, \text{“VLDB04”}), \text{repN}(7, \langle \text{author} \rangle X^{25} \langle / \text{author} \rangle^{24})\}$ applicable on the document in Fig. [1](#). The inverse of Δ , obtained by inverting each single operation independently is $\Delta^{-1} = \{op_1, op_2, op_3, op_4\}$ where $op_1 = \text{ins}^\swarrow(4, \langle \text{name} \rangle \text{EDBT04 W...}^6 \langle / \text{name} \rangle^5)$, $op_2 = \text{repV}(6, \text{“EDBT04 W...”})$, $op_3 = \text{del}(24)$, $op_4 = \text{ins}^\swarrow(4, \langle \text{author} \rangle \text{B.Catania}^8 \langle / \text{author} \rangle^7)$. Δ^{-1} exhibits two problems: (i) the overridden operation $\text{repV}(6, \text{“VLDB04”})$ has been inverted as op_2 , that is both unnecessary (the value of node 6 is restored by op_1) and not applicable, as node 6 does not belong to any document $D' \in \mathcal{O}(\Delta, D)$; (ii) the order of the restored nodes 5 and 7 may not be the one in the original document.

To guarantee the first two properties, overridden operations in Δ should be removed. For the last property, special treatment should be devoted to operations del and repN when their targets are adjacent siblings. In this case, insertion operations of the same kind with the same target might not preserve the insertion order. To obtain the correct order, repN and del operations on adjacent siblings should be grouped together.

Definition 3 (Removal Group). Given a PUL Δ , we denote as a removal group of Δ a non-empty ordered sublist $S = [op_1, \dots, op_n]$ of Δ s.t.

- $o(op_i) \in \{\text{repN}, \text{del}\}, 1 \leq i \leq n$ and $t(op_1) \triangleleft_s t(op_2) \triangleleft_s \dots \triangleleft_s t(op_n)$,
- $\{op_1, \dots, op_n\}$ is maximal, that is, $\nexists S' \supset \{op_1, \dots, op_n\}$ s.t. S' is a removal group of Δ .

Example 5. Consider the PUL $\Delta = \{\text{del}(7), \text{ren}(5, \text{“title”}), \text{repN}(5, \langle \mathbf{a} \rangle X \langle / \mathbf{a} \rangle), \text{del}(11)\}$ on the document in Fig. [1](#). The removal groups are $[\text{repN}(5, \langle \mathbf{a} \rangle X \langle / \mathbf{a} \rangle), \text{del}(7)]$ and $[\text{del}(11)]$.

{ins<(4, <name>EDBT04 W...⁶ </name>⁵ <author>B.Catania⁸ </author>⁷), del(24)}. In this case, differently from what happened in Example 4, the overridden operation `repV(6, "VLDB04")` has not been inverted and the order of the restored nodes 5 and 7 is preserved.

Proposition 2 (Correctness of the Inversion Rules). *Let Δ be a PUL applicable on a document D . The PUL Δ^{-1} obtained through the application of the inversion rules in Fig. 2 is an inverse of Δ according to Definition 2* △

Proof Sketch. Consider a PUL Δ applicable on a document D and the inverse Δ^{-1} generated by the inversion rules. The proof of this proposition requires that: (i) Δ^{-1} is applicable on each document in $\mathcal{O}(\Delta, D)$, (ii) no pairs of incompatible operations are generated, (iii) no partially overridden operation either in Δ or Δ^{-1} exists, (iv) nodes removed by Δ are placed back in the correct positions by Δ^{-1} , (v) Δ^{-1} is deterministic. The proof of (i) is a straightforward consequence of the removal of overridden operations and of the operations definition. (ii) can be proved considering the algorithm definition and the applicability of Δ on D , while (iii) considering the operations definition. The proof of the other points comes directly from the definition of the algorithm, specifically (iv) is ensured by the class G rules, while (v) follows from the analysis of the generated operations.

3.3 Inversion Algorithm

Algorithm 1 presents an efficient procedure for computing the inverse of a PUL Δ . The following functions are exploited in the algorithm: *applyLocalOverrideRules*(Δ) to apply the reduction rules O1 and O2 in Fig. 2 on Δ ; *applySInversionRules*(Δ, D) and *applyGInversionRules*(Δ, D) to apply the inversion rules of class S and G.

The inversion algorithm works as follows. An empty PUL Δ^{-1} is first initialized, then operations in Δ are ordered according to the pre-order traversal of their target nodes and grouped together. Note that, if an overriding operation *op* is present in a group, the groups of the overridden operations immediately follow that of *op*. Then, for each group of operations Δ_{v_i} on a node v_i , the algorithm performs the following steps. (i) It checks whether the operations Δ_{v_i} are overridden by operations specified on an ancestor of v_i , in this case they are discarded (this corresponds to the application of rules O3 and O4), otherwise, the *applyLocalOverrideRules* function is applied on the operations of v_i . (ii) Whenever in Δ_{v_i} there is an operation *op* that may override operations in the subtree rooted at v_i , v_i is stored along with the relevant information about *op*. (iii) The rules of class S are applied on the remaining operations on v_i through *applySInversionRules*, updating Δ_{v_i} and adding the computed inverses to Δ^{-1} .

When all the partitions have been processed, the remaining operations in Δ are all and only those that belong to a removal group and each removal group is composed of operations that are contiguous in Δ . Function *applyGInversionRules* can thus be efficiently applied on Δ , adding the inverses to Δ^{-1} .

Proposition 3 (Complexity). *Let Δ be a PUL applicable on a document D , removing r nodes. The complexity of the Algorithm 1 is $\mathcal{O}(n \log(n) + sn + r)$ where n is the size of Δ and s the cost of identifying the left sibling/parent of a node in D .* △

Algorithm 1. Inversion

Require: A PUL Δ applicable on a document D

1. $\Delta^{-1} = \emptyset$;
2. $(o, v_o) = (\perp, \perp)$;
3. let $(\Delta_{v_1}, \dots, \Delta_{v_n})$ be the partition of Δ according to the preorder of their target node;
4. **for** $i = 1$ **to** n **do**
5. **if not** $((o = \text{pAttr} \wedge v_i \parallel_d^{-a} v_o) \vee (o = \text{rAttr} \wedge v_i \parallel_d v_o))$ **then**
6. $\Delta_{v_i} = \text{applyLocalOverrideRules}(\Delta_{v_i})$;
7. $\text{delOp} = \{o \mid o \in \Delta_{v_i}, \tau(v_i) = \mathbf{e}, o(o) \in \{\text{repC}, \text{repN}, \text{del}\}\}$;
8. $(o, v_o) = \begin{cases} (\text{pAttr}, v_i) & \text{if } \text{repC} \in \text{delOp} \\ (\text{rAttr}, v_i) & \text{if } \text{repN} \in \text{delOp} \vee \text{del} \in \text{delOp} \end{cases}$
9. $\Delta^{-1} = \Delta^{-1} \cup \text{applySInversionRules}(\Delta_{v_i}, D)$
10. **end if**
11. **end for**
12. $\Delta^{-1} = \Delta^{-1} \cup \text{applyGInversionRules}(\Delta, D)$

Ensure: Δ^{-1} is an inverse of Δ on D according to Definition 2

Proof Sketch. The algorithm first requires to sort the PUL according to the pre-order traversal of their target nodes, which can be performed in $\mathcal{O}(n \log n)$, employing the labeling information and standard algorithms. Overridden operation removal then requires a single scan of the PUL, thus $\mathcal{O}(n)$. The application of inversion rules of class S requires to consider each remaining operation and might identify up to $n + r$ operations, ($\mathcal{O}(n + r)$). Finally, the application of the inversion rules of class G might require, for each operation, to determine the left sibling/parent of the operation target node in D . Assuming this cost s , the cost is $\mathcal{O}(sn)$, and, thus, the algorithm complexity is $\mathcal{O}(n \log(n) + sn + r)$.

4 Completed PULs

The approach discussed in the previous section, starting from a PUL Δ , identifies another PUL Δ^{-1} which reverts the effects of Δ . An alternative approach is to extend the operations presented in Table 2 with auxiliary information to allow their inverse (i.e., backward) application. These extended operations are reported in the last column of Table 2 and are referred to as *completed operations*. The PUL obtained from Δ by replacing its operations with the corresponding completed operations is named *completed PUL* and denoted by Δ^{\leftrightarrow} . Completed PULs can be applied either forward (obtaining the same effect of the original PUL Δ) or backward (obtaining the same effect of one of its inverses). This approach does not require the explicit identification of an inverse PUL and avoids to access the document. Indeed, all required data is already contained in the completed PUL, which can be applied in streaming in both directions.

The forward application of a completed PUL simply consists in the streaming application of the corresponding PUL, i.e., ignoring additional information included in completed operations. The definition of obtainable documents is trivially extended to completed PULs: $\mathcal{O}(\Delta^{\leftrightarrow}, D) = \mathcal{O}(\Delta, D)$. The backward application of a completed PUL can be performed by applying a different set of transformations on event sequences.

Specifically: *remove* a node and its subtree from the sequence (for removing any inserted node); *restore* a subtree in its original position in the sequence (for restoring any removed subtree); *rename* a node (for inverting *ren* operations); *change value* to a node (for inverting *repV* operations). Consider a completed PUL Δ^{\leftrightarrow} applicable on a document D , and a document $D' \in \mathcal{O}(\Delta^{\leftrightarrow}, D)$. The algorithm for the backward application of Δ^{\leftrightarrow} on D' identifies a set of transformations for Δ^{\leftrightarrow} and applies them on the event sequence corresponding to D' , obtaining the sequence of events corresponding to D . Note that no transformation must be applied on a restored subtree, since it is already restored as in the original document.

Example 7. Consider the completed PUL $\Delta^{\leftrightarrow} = \{\text{del}(5, \langle \text{name} \rangle \text{EDBT04 W...}^6 \langle / \text{name} \rangle^5), \text{repV}(6, \text{"VLDB04", "EDBT04 W..."}), \text{repN}(7, \langle \text{author} \rangle \text{X}^{25} \langle / \text{author} \rangle^{24}, \langle \text{author} \rangle \text{B.Catania}^8 \langle / \text{author} \rangle^7)\}$ applicable to the document in Fig. 11. To backward apply Δ^{\leftrightarrow} , the inserted subtree (rooted at node 24) must be removed, while the removed subtrees (rooted at node 5 and 7) must be reintroduced. The inverse application of an overridden operation poses no problem, since the corresponding transformation has no effect. For instance, the inversion *repV* requires to change back the value of node 6, but node 6 will not be considered, as it is restored by another transformation.

Algorithm 2 realize the backward application of a completed PUL Δ^{\leftrightarrow} , applicable on a document D , on a document $D' \in \mathcal{O}(\Delta^{\leftrightarrow}, D)$, identifying the sequence of events corresponding to D . Given Δ^{\leftrightarrow} and the sequence of events E' corresponding to D' , the algorithm produces the sequence of events E corresponding to D . The *emit* e_1, \dots, e_n directive is employed to indicate that the events e_1, \dots, e_n are generated. For the sake of conciseness, we denote the node, name, and value components of an event e , as *e.v*, *e.n*, and *e.s*, respectively. Function *Events* is used to associate a set of subtrees, considered according to the pre-order traversal of their roots in D , with the corresponding sequence of events. The algorithm works as follows. First, the sets I and R of the nodes inserted (that must be removed) and removed (that must be restored) by Δ^{\leftrightarrow} are identified. Then, the algorithm processes the document. The *sDoc* event is simply emitted back when encountered, while the other events are distinguished and the transformation applied. When the *eDoc* event is encountered, in case the root of D is present in R (denoted R_{root}), it is restored before emitting back *eDoc*. For each other event $e \in E'$, the original name and value of node *e.v* are determined (undefined if *e.v* does not have a name/value property), then one of the following steps is performed: (i) In case e is *sElem* or *text*: if the parent of *e.v* is not being removed, any removed preceding siblings of *e.v* in R (denoted R_{\leftarrow}) is emitted. Afterwards, if node *e.v* should not be removed e is emitted with the original name and value of *e.v*, (followed by the attributes of *e.v* in R , denoted R_{attr} , if e is a *sElem*). (ii) In case e is *eElem*: if *e.v* should not be removed, the children of node *e.v* in R (denoted R_{\searrow}) are emitted, followed by e with the original name and value of *e.v*. (iii) In case e is *attr*: e is emitted with the original name and value of *e.v*, unless *e.v* has to be removed. To avoid restoring multiple times the same subtree, subtrees are removed from R when restored.

Proposition 4 (Correctness). *Let Δ^{\leftrightarrow} be a completed PUL applicable on a document D , and let D' be a document in $\mathcal{O}(\Delta^{\leftrightarrow}, D)$. The application of Algorithm 2 on Δ^{\leftrightarrow} and D' produces D .*

Algorithm 2. Completed PUL streaming backward application

Require: A completed PUL Δ^{\leftrightarrow} applicable on a document D , the sequence of events $[e_1, \dots, e_n]$ for a document $D' \in \mathcal{O}(\Delta^{\leftrightarrow}, D)$

1. $I = \{V(p(op)) \mid o(op) \in \{\text{ins}, \text{repN}, \text{repC}\} \wedge op \in \Delta^{\leftrightarrow}\}$ be the nodes inserted by Δ^{\leftrightarrow}
2. $R = \{T(t(op)) \mid o(op) \in \{\text{del}, \text{repN}\} \wedge op \in \Delta^{\leftrightarrow}\} \cup \{T(\gamma(t(op))) \mid o(op) = \text{repC}, op \in \Delta^{\leftrightarrow}\}$ be the subtrees removed by Δ^{\leftrightarrow}
3. **for** $i = 1$ **to** n **do**
4. **if** $e_i = \text{sDoc}()$ **then**
5. emit e_i
6. **else if** $e_i = \text{eDoc}()$ **then**
7. $R_{\text{root}} = \begin{cases} T & \text{if } \exists T \in R \text{ s.t. } T \text{ is the document root} \\ \emptyset & \text{otherwise} \end{cases}$
8. emit $\text{Events}(R_{\text{root}}, e_i)$
9. **else**
10. $\bar{n} = \begin{cases} n' & \text{if } \exists \text{ren}(e_i.v, n, n') \in \Delta^{\leftrightarrow} \\ e_i.n & \text{otherwise} \end{cases} \quad \bar{s} = \begin{cases} s' & \text{if } \exists \text{repV}(e_i.v, s, s') \in \Delta^{\leftrightarrow} \\ e_i.s & \text{otherwise} \end{cases}$
11. **if** $e_i = \text{sElem}(v, n)$ **then**
12. $R_{\leftarrow} = \begin{cases} \{T_1, \dots, T_n\} & \text{if } \exists [T_1, \dots, T_n] \text{ removal group in } R \text{ s.t. } \mathcal{R}(T_n) \blacktriangleleft_s v \\ \emptyset & \text{otherwise} \end{cases}$
13. **if** $\nexists v' \in I$ s.t. $v /_c v'$ **then** emit $\text{Events}(R_{\leftarrow})$ **end-if**
14. $R_{\text{attr}} = \{T_1, \dots, T_n \mid \forall i, 1 \leq i \leq n, T_i \in R, \mathcal{R}(T_i) /_a v\}$
15. **if** $v \notin I$ **then** emit $\text{sElem}(v, l, \bar{n}), \text{Events}(R_{\text{attr}})$ **end-if**
16. $R = R \setminus (R_{\leftarrow} \cup R_{\text{attr}})$
17. **else if** $e_i = \text{eElem}(v, n)$ **then**
18. $R_{\searrow} = \begin{cases} \{T_1, \dots, T_n\} & \text{if } \exists [T_1, \dots, T_n] \text{ removal group in } R \text{ s.t. } \mathcal{R}(T_n) /_c v \\ \emptyset & \text{otherwise} \end{cases}$
19. **if** $v \notin I$ **then** emit $\text{Events}(R_{\searrow}), \text{eElem}(v, \bar{n})$ **end-if**
20. $R = R \setminus R_{\searrow}$
21. **else if** $e_i = \text{attr}(v, n, s)$ **then**
22. **if** $v \notin I$ **then** emit $\text{attr}(v, \bar{n}, \bar{s})$ **end if**
23. **else if** $e_i = \text{text}(v, s)$ **then**
24. $R_{\leftarrow} = \begin{cases} \{T_1, \dots, T_n\} & \text{if } \exists [T_1, \dots, T_n] \text{ removal group in } R \text{ s.t. } \mathcal{R}(T_n) \blacktriangleleft_s v \\ \emptyset & \text{otherwise} \end{cases}$
25. **if** $\nexists v' \in I$ s.t. $v /_c v'$ **then** emit $\text{Events}(R_{\leftarrow})$ **end-if**
26. $R = R \setminus R_{\leftarrow}$
27. **if** $v \notin I$ **then** emit $\text{text}(v, \bar{s})$ **end if**
28. **end if**
29. **end if**
30. **end for**

Ensure: The sequence of events emitted models the document D .

Proof Sketch. The inverse application of a completed PUL Δ^{\leftrightarrow} must (i) remove all inserted nodes; (ii) restore the original value and name of nodes, and (iii) insert back in the document any removed node in its original position. When the updated document is processed any node that has been inserted by Δ^{\leftrightarrow} (the nodes in I) is not emitted (removed). Otherwise the node is emitted back with its original value and name, which is retrieved considering the ren and repV operations in Δ^{\leftrightarrow} . In this process, the node labels and the set of removed subtrees R are used to determine if any node must be

restored between two encountered nodes, ensuring that the original document nodes are restored in their original position. Therefore, since the three properties are met by Algorithm 2, we are guaranteed that the original document is produced.

Proposition 5 (Complexity). *Let Δ^{\leftrightarrow} be a completed PUL of size n applicable on a document D , let D' be a document in $\mathcal{O}(\Delta^{\leftrightarrow}, D)$ composed of d nodes, and let i and r be the number of nodes inserted and removed by Δ^{\leftrightarrow} , respectively. The complexity of Algorithm 2 is $\mathcal{O}(n \log n + d + i + r)$.*

Proof Sketch. The identification of the nodes to be removed and the original name and value of updated nodes can rely on a hash-table, which requires $\mathcal{O}(n)$ for its initialization and $\mathcal{O}(1)$ for retrievals. For what concerns the restoration of nodes, a more efficient representation of R is a list ordered according to the pre-order traversal of the subtrees roots, with cost $\mathcal{O}(n \log n)$. Moreover, since events are generated according to a pre-order visit of the tree, identifying whether there is a subtree to restore, requires to check only the first element in the list, with cost $\mathcal{O}(1)$, assuming that subtrees removed multiple times are pruned during the inverse application. Assessing whether the parent of a node has been removed requires constant time, provided that this information is stored in a state variable. Thus, since all nodes in the updated document need to be processed and up to r nodes need to be restored, the complexity of the algorithm is $\mathcal{O}(n \log n + d + i + r)$.

5 Evaluation

In this section we discuss some experiments we have conducted by means of the extended Qixx library that is able to generate and process both PULs and completed PULs represented as XML documents containing the serialization of each operation along with the identifiers and labels of the target nodes. Our test machine uses an Intel I5 760 processor, 16GB of RAM, and runs the Sun JDK v.1.6.20.

To assess the computational costs of the inversion operator and contrast the peculiar advantages of PULs and completed PULs, we exploit documents of various sizes, ranging from 16MB to 256MB, produced by means of the XMark data generator. Node identifiers and labeling have been stored within the corresponding documents through an encoding of their XDM structure. On such documents, synthetic XQU expressions and their corresponding PULs/completed PULs have been generated with a varying number of operations that are equally distributed among the operation types.

Starting from an XQU expression the generation of a PUL Δ /completed PUL Δ^{\leftrightarrow} requires to load the whole document in memory before the actual evaluation of the XQuery Update expression. When the expression is evaluated, generating a completed PUL has the additional cost of retrieving and storing within the PUL itself all the modified values and removed nodes. Analogously, the generation of PUL inversion requires to load the whole document in memory. The application of the so generated PULs, by contrast, requires to load the whole PUL in memory and then update the corresponding document through a single scan identifying a new document. While Δ and Δ^{-1} serializations contain only the strictly required information for their application, the serialization of Δ^{\leftrightarrow} contains extra information that is discarded depending on the direction (forward or backward) of application.

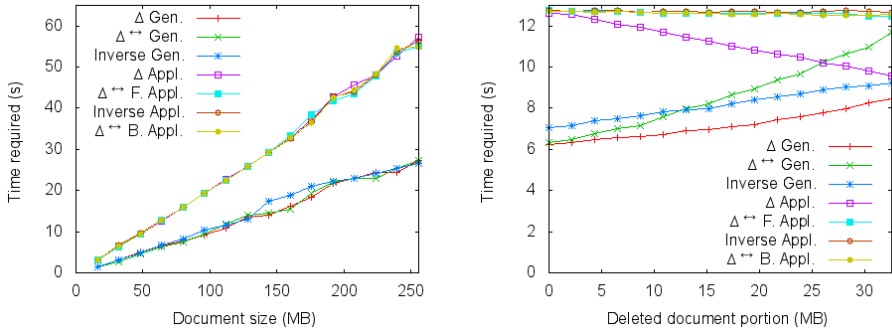


Fig. 3. Experiments

Given a synthetic XQU expression, we first investigated the correlation between the size of the document and the time required for: (i) generating the corresponding PUL Δ /completed PUL Δ^{\leftrightarrow} ; (ii) computing the inverse Δ^{-1} of Δ ; (iii) applying Δ , Δ^{\leftrightarrow} , Δ^{-1} and backward applying Δ^{\leftrightarrow} . In all cases, the portion of the document being inserted or removed is approximately 1MB. Results for this experiment are reported on the left side of Fig. 3. Moreover, we consider a 64MB document and we vary the amount of information removed or replaced (ranging from 0% to 50%). Results of the second experiment are reported in the right side of Fig. 3. The experiments indicate that the cost of generating or applying a PUL Δ^* (either Δ , Δ^{\leftrightarrow} , or Δ^{-1}) is dominated by the number of nodes analysed. Specifically, when a PUL Δ or Δ^{\leftrightarrow} is applied, the number of nodes in the original document, those in the updated document and those in the PUL itself contribute to the running time of PUL application. Analogously, for the application of Δ^{-1} , the number of nodes in the original document and those in Δ and Δ^{-1} affect the running time. Thus, when the portion of the document being inserted or deleted is not relevant, PULs generation and application time are not influenced by Δ^* , while applications require almost twice the time of generations. By contrast, when the size of the deleted portions of the document increases, completed PULs become less efficient, as the forward application time is unchanged and its generation time increases. Similarly, when the number of nodes inserted by a PUL is high, PUL backward application time remains unchanged, and its generation time increases, even if to a lesser extent.

6 Related Work

The paper relies on the approach to update execution proposed in [3] which proposes the decoupling of PUL production from PUL evaluation as an execution model for XML updates expressed through XQU expressions in distributed contexts. The need of more flexible mechanisms for update handling entails the development of suitable mechanisms for reasoning on updates before actually apply them, and specifically for composing as well as reverting them. In [3], PUL operators for composing updates (integration and aggregation) as well as for devising a compact representation (reduction) are investigated, but inversion is not addressed.

The notion of completed PULs is inspired by completed deltas proposed by [11] in the context of XML versioning. The goal of the two notions is the same, in that they both aim at having a compact representation of changes that can be applied either forward (to actually apply them) or backward (to revert their effects). However, completed deltas are obtained by comparing two document versions (through diff algorithms) and do not represent the effects of XQU expressions. As a consequence, both the set of primitive operations and the associated semantics are different. A peculiar aspect in inverting updates expressed as PUL is indeed related to the XQU snapshot semantics by which a PUL is an unordered list of operations, that have to be applied on documents according to some precedence among operators prescribed by [15] and formalized in a five stage semantics in [1]. The inversion mechanism proposed in the paper is designed according to that semantics. By contrast, completed deltas refer to sequences of non-conflicting operations.

Though our approach is not specifically targeted to a transactional context, the ability of reverting the effects of XML updates could also be useful in that setting. The notion of compensating transaction as a transaction that semantically undoes the partial effects of a transaction without performing cascading aborts of dependent transactions, restoring the system to a consistent state, has been proposed in the context of long-lived transaction [6,7,10] and is particularly relevant in the context of workflows and web services [13]. These types of compensation range from traditional undo, at one extreme, to application-dependent, special-purpose compensating transactions, at the other extreme. XML transactions have been investigated in [5,8,9] but the focus was on isolation levels and lock mechanisms. An approach to atomicity for XML transactions relying on statement undos is proposed in [2]. They consider the update operations in a PUL as separate transactions and discuss how individual operations can be undone. However, interactions among different operations in a PUL, e.g., overriding, neither on the same node nor on nodes bound by hierarchical relationships in the tree, are considered.

7 Concluding Remarks

In this paper we considered updates on XML documents expressed through XQU expressions and the corresponding dynamic model of updates based on PULs. We investigated the issue of reverting the effects of an update expression, referring to the case in which PUL production is decoupled from their application. Two alternative approaches have been considered: the first one is the inversion of a PUL through an inversion operator. The second one is the extension of the PUL model (completed PULs) so that the required information for both a forward and a backward (inverse) application are included. We presented, discussed, and implemented in the Qizx library the algorithms for both inverting a PUL and for the streaming backward application of a completed PUL. Finally, we contrasted the two proposed approaches through an experimental evaluation. As future work we plan to investigate the correlation between the inversion operators presented in this paper with the operations proposed in [3] in order to identify an algebra on PULs. Moreover, we wish to tailor the developed operator in specific contexts like versioning, transaction, and cloud. Finally, in the current setting, labeling information is stored within the document. This has the effect of increasing

the document size of three times. This issue could be solved by considering a shredded representation of the document and, as future work, we wish to explore this possibility.

References

1. Benedikt, M., Cheney, J.: Semantics, Types and Effects for XML Updates. In: Gardner, P., Geerts, F. (eds.) *DBPL 2009*. LNCS, vol. 5708, pp. 1–17. Springer, Heidelberg (2009)
2. Biswas, D., Jiwane, A., Genest, B.: Atomicity for XML Databases. In: Bellahsene, Z., Hunt, E., Rys, M., Unland, R. (eds.) *XSym 2009*. LNCS, vol. 5679, pp. 180–187. Springer, Heidelberg (2009)
3. Cavalieri, F., Guerrini, G., Mesiti, M.: Dynamic Reasoning on XML Updates. In: *EDBT*, pp. 165–176. ACM Digital Library (2011)
4. Chien, S.-Y., Tsotras, V.J., Zaniolo, C.: Efficient Schemes for Managing Multiversion XML Documents. *VLDB J.* 11(4), 332–353 (2002)
5. Dekeyser, S., Hidders, J., Paredaens, J.: A Transaction Model for XML Databases. *World Wide Web* 7(1), 29–57 (2004)
6. Elmagarmid, A.K. (ed.): *Database Transactional Models for Advanced Applications*. Morgan Kaufmann, San Francisco (1992)
7. Garcia-Molina, H., Salem, K.: Sagas. In: *SIGMOD Conference*, pp. 249–259. ACM Press, New York (1987)
8. Grabs, T., Böhm, K., Schek, H.-J.: XMLTM: Efficient Transaction Management for XML Documents. In: *CIKM*, pp. 142–152 (2002)
9. Helmer, S., Kanne, C.-C., Moerkotte, G.: Evaluating Lock-based Protocols for Cooperation on XML Documents. *SIGMOD Record* 33(1), 58–63 (2004)
10. Korth, H.F., Levy, E., Silberschatz, A.: A Formal Approach to Recovery by Compensating Transactions. In: *VLDB*, pp. 95–106 (1990)
11. Marian, A., Abiteboul, S., Cobena, G., Mignet, L.: Change-Centric Management of Versions in an XML Warehouse. In: *VLDB*, pp. 581–590 (2001)
12. O’Connor, M.F., Roantree, M.: Desirable Properties for XML Update Mechanisms. In: *Updates in XML EDBT/ICDT Workshop* (2010)
13. Peltz, C.: Web Services Orchestration and Choreography. *Computer* 3(10), 46–52 (2003)
14. PIXwere Ltd. *QIZX*. An Open-source XQuery Processor (2010)
15. W3C. *XQuery Update Facility 1.0* (June 2009)

TraCX: Transformation of Compressed XML

Stefan Böttcher, Rita Hartel, and Sebastian Stey

University of Paderborn, Computer Science, Fürstenallee 11, 33102 Paderborn, Germany
{stb@,rst@,sebstei@mail.}uni-paderborn.de

Abstract. While the volume of XML data and sometimes even the processing time of XML data can be reduced by using XML compression and storing, processing, and transferring compressed XML instead of uncompressed XML, a transformation of the transferred XML data into the receiver's XML format via XQuery cannot be performed on the compressed XML data directly. Instead, XQuery transformation of compressed XML data requires a prior decompression. In this paper, we present a generic approach to transform compressed or uncompressed XML representations that support basic navigation and update as well as optional copy functionalities. In a series of experiments, we have shown that using our approach to transform compressed XML is not only faster than the indirect approach via decompression, XQuery transformation, and recompression, but also that our approach transforms compressed XML as efficient as other XQuery evaluators transform uncompressed XML only.

Keywords: XML compression, XQuery, XML transformation.

1 Introduction

1.1 Motivation

Due to its flexible structure, XML enjoys increasing popularity as a data exchange format and as an intermediate storage format, e.g. in production chains where XML data is exchanged, transformed and eventually further processed sequentially by multiple participants. But the overhead caused by this flexible structure forms the biggest disadvantage of using XML as a data exchange or storage format. Transferring and storing a compressed XML representation instead of uncompressed XML might help to solve this problem. However, when XML files have to be shared with other parties, often the XML structure needs to be adapted to match the XML data structure on the receiver's side. A typical approach to adapt uncompressed XML files is to transform them via XQuery [1] into the format that is required on the receiver's side. But when exchanging compressed XML, the decompression prior to the XQuery transformation and, after the XQuery transformation, the recompression of transformed XML data needed for compact storage and for further transfer of compressed XML data may be a significant overhead. Instead, the XQuery transformation of compressed data appears to be a promising alternative. However, up to now, XQuery transformation of compressed XML data has been very limited, i.e., up to now, there has not been any generic approach to XQuery evaluation applicable to all XML compression techniques. Instead, most XML compression techniques do not support XQuery at all, and

the few exceptions have at least one of the following weaknesses. First, XQueC [2] suffers from a weaker XML compression and is designed to output uncompressed XML only. Second, for XQuery evaluation on top of Bisimulation [3], the transformation of XQuery is limited to a small XQuery subset that only supports XPath expressions consisting of child-axis location steps only. In comparison, we propose a generic approach to transform compressed XML documents that is applicable to various XML compression techniques that support a basic set of navigation and update operations on compressed XML data. Furthermore, our transformation on compressed XML outperforms the indirect approach via decompression, XQuery transformation and recompression and works as efficiently as other XQuery evaluators that transform uncompressed XML.

1.2 Contributions

In this paper, we present a generic approach to transforming compressed or uncompressed XML representations that supports a subset of XQuery (c.f. Section 1.3) and that combines the following properties:

- It allows transforming each XML representation, and especially each compressed XML representation, that supports the basic binary navigation operations ‘navigation to the first-child’, ‘navigation to the next-sibling’, and ‘navigation to the end-tag of the parent’, and supports the basic update operations ‘appending a first-child to the current end of the document’, ‘appending a next-sibling to the current end of the document’ and ‘closing the current node’, i.e., adding an end tag and continuing with the next-sibling or the end-tag of the parent node.
- Whenever an XML compression technique supports these basic functionalities directly on a compressed representation $c(X)$ of an XML document X , our approach allows transforming $c(X)$ into another compressed XML file $c(X')$ corresponding to the new XML structure X' of the transformed XML document X .
- Whenever an XML representation not only supports these basic functionalities, but supports, for example, evaluating XPath queries efficiently or copying compressed sub-trees from a compressed source document into a compressed target document without having to decompress the sub-tree, these capabilities are used by our approach instead. Unnecessary decompression and recompression of the compressed document is avoided.

We have implemented and evaluated our transformation approach for uncompressed XML in form of a SAX-based XML representation and for compressed XML in form of Succinct compression [4]. These are examples of XML representations that can be combined with our approach. All other XML representations that fulfill the above given requirements can work with our approach as well.

1.3 Query Language

The transformation language used in our approach is a subset of XQuery [1]. It can be defined by the following EBNF grammar, where XPATH is a relative coreXPath expression containing only the forward axes self, attribute, text, child, descendant, descendant-or-self, and following-sibling, and where NAME , VARIABLE ,

COMPRESSIONTYPE, and FILEPATH are Strings, and SEQ is a sequence of Strings. The terminal symbol ‘;’ represents a carriage return.

```

XMLTREE ::= '<' NAME '>'; XML* '</' NAME '>'
XML ::= XMLTREE; | FLR;
FLR ::= '{'; (DEFINITION;)* OUTPUT; '}'
DEFINITION ::= FOR | LET
FOR ::= 'for $' NAME 'in' EXPRESSION
LET ::= 'let $' NAME ':=' EXPRESSION
EXPRESSION ::= DOC '/' XPATH | '$' VARIABLE '/' XPATH | SEQ
DOC ::= 'doc("` FILEPATH `", "` COMPRESSIONTYPE `")'
OUTPUT ::= 'return'; XMLRET
XMLRET ::= XMLRETTREE; | '{$' VARIABLE '}'
XMLRETTREE ::= '<' NAME '>'; XMLRET* '</' NAME '>'
    
```

1.4 Example Being Used in This Paper

In this paper, we use a short example for illustrating the ideas. Fig. 1 (a) shows the source document of our example that contains a list of countries and rivers for a continent and that contains a list of cities for each country. Fig. 1 (b) shows the target document in which the root element is the element mainland (corresponding to the element continent of Fig. 1 (a)). Furthermore, the country is renamed to nation and the rivers are removed. Fig. 1 (c) shows the transformation instruction that allows transforming the source document into the target document.

(1) <cont>	(1) <mainland>	(1) <mainland>
(2) <country>	(2) <nation>	(2) {
(3) <city>C1</city>	(3) <city>C1</city>	(3) for \$country in
(4) <city>C2</city>	(4) <city>C2</city>	(4) doc(„file“)/cont/country
(5) </country>	(5) </nation>	(5) let \$city := \$country/city
(6) <river>		(6) return
(7) <name>R1</name>		(7) <nation>
(8) </river>		(8) { \$city }
(9) <country>	(6) <nation>	(9) </nation>
(10) <city>C3</city>	(7) <city>C3</city>	(10) }
(11) </country>	(8) </nation>	(11) </mainland>
(12) </cont>	(9) </mainland>	
(a)	(b)	(c)

Fig. 1. (a) source document, (b) target document, (c) transformation instructions

1.5 Paper Organization

This paper is organized as follows: Section 2 summarizes the basic idea and the fundamental concepts underlying our approach to transform compressed XML documents followed by a detailed discussion of the operations needed and how they can be performed on compressed and on uncompressed XML. The third section outlines some of the experiments that compare our approach with the existing XQuery evaluator Zorba. Section 4 gives an overview of related work and is followed by the Summary and Conclusions.

2 Our Solution

2.1 Basic Idea

When evaluating XQuery queries over documents given in a compressed or uncompressed XML representation, there are only few parts of the evaluation process that are specific for the chosen XML representation, whereas most parts are generic and can be implemented by our approach directly and independently of the chosen XML representation.

The specific XML representation has to provide the evaluation of absolute and relative XPath expressions within the source document. Furthermore, the specific XML representation has to provide inserting new elements as the first-child or as the next-sibling of the current context node, and copying sub-trees from the source document to the target document.

But even these tasks contain generic aspects. The evaluation of XPath expressions can be reduced to navigation via the basic binary axes first-child, next-sibling, and end-of-parent and the retrieval of the label of the current context node. Copying sub-trees can be reduced to navigation via the same basic binary axes within the source document and inserting new elements as first-child or next-sibling into the target document. Nevertheless, our evaluation has shown that our approach runs faster, if XPath evaluation and copying sub-trees is optimized according to the specific representation instead of using the generic solution via navigation and updates.

To execute a transformation instruction, we parse the transformation instruction line by line and interpret the lines as follows: Whenever there is a call to an element constructor (as e.g. `<mainland>`), we add the constructed element to the target document as first-child or as next-sibling of the current context element. Whenever there is a for-statement FS, we evaluate the absolute or relative XPath expression of FS and execute the content of FS for each result of the XPath expression. Whenever there is a let-statement LS, we evaluate the absolute or relative XPath expression of LS, concatenate the results of the XPath expression to a single sequence and bind the whole sequence to the variable given within LS. In order to minimize the number of passes through the source document, we do not evaluate the XPath expressions of the for- and let-statements line by line as they appear in the transformation instruction, but we first collect all XPath expressions and evaluate all of them in parallel considering the interdependencies within the queries. Whenever there is a call to a variable `$v` within the return-statement, i.e., whenever it is required to output the content of `$v` to the target document, we copy the content of `$v` from the source document to the target document.

Most tasks that have to be performed in order to transform a source document according to an XQuery transformation into a target document are independent of the XML representation used and can be solved generically. But in order to be transformable via our approach, an XML representation has to provide either the navigation via first-child, next-sibling and end-of-parent, or – in order to be sure that the transformation requires 2 passes only – it has to provide the evaluation of several absolute and relative XPath expressions in parallel. If an XML representation does only allow the sequential evaluation of several XPath expressions, sequential evaluation can be used as well, but then the transformation requires more than just 2 passes. Furthermore, the

representation has to provide the capability to insert a new element as the first-child or the next-sibling of the current context element, and in addition it might optionally provide the capability to copy whole sub-trees from a source to a target document.

In the remainder of this section, we first explain for each task, i.e. XPath evaluation and copying sub-trees from the source document to the target document, how this task is solved in the generic part, and then discuss for two XML representations – uncompressed XML on the one hand and Succinct Compression [4] on the other hand – how the representation specific part is solved for these XML representations.

2.2 XPath Evaluation

Generic XPath Evaluation. We follow the idea of [5] and use an automata-based approach to XPath evaluation. We reduce the set of axes to the basic binary axes first-child, next-sibling, end-of-parent, and self::label, where label is the label of any XML element or ‘*’. We assume that each XML representation produces binary XML events of the form first-child, next-sibling, end-of-parent or self::label when passing through the XML representation. That is, when we navigate from the current context node to its first-child that carries the label ‘lab’, this sends two events to the automaton: a first-child::* event followed by the event self::lab.

For each XPath forward axis, we provide an atomic automaton using the basic binary axes. Fig. 2 shows the atomic XPath automata for the location steps (a) child::e and (b) descendant::f and (c) a combined automaton for the XPath expression /child::e/descendant::f.

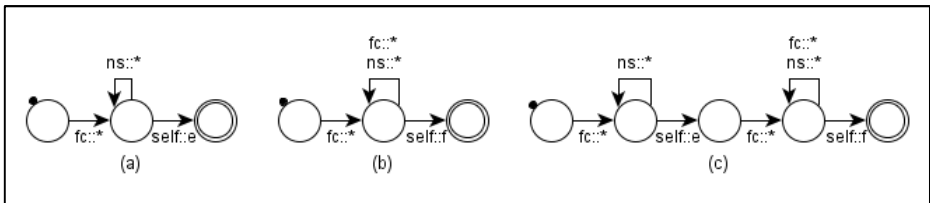


Fig. 2. XPath automata for (a) child::e (b) descendant::f, and (c) /child::e/descendant::f

According to the order of the location steps within the XPath expression XP, the atomic automata for all location steps are concatenated to form the XPath automaton of the whole XPath expression XP. As these automata only contain forward axes, they allow evaluating the XPath expression within a single pass through the document. More details on the automata-based XPath evaluation are described in [5].

In contrast to [5], which handles the evaluation of absolute queries only, we have to handle absolute and relative XPath expressions. In order to evaluate all XPath expressions XPA1, ..., XPA_n, XPR1, ..., XPR_m occurring in the given XQuery expression, where each XPA_i is an absolute XPath expression and each XPR_j is a relative XPath expression, we generate one XPath automaton for each absolute or relative XPath expression XP. When parsing of the XML representation starts, only the automata for the absolute expressions XPA1, ..., XPA_n are active and consume the navigation events of the form first-child, next-sibling, self::label, and end-of-parent that

are created by navigating through the XML representation. Whenever a result of an XPath expression is reached, the result is bound to the XQuery variable $\$v$ assigned to this XPath expression in the transformation instruction, and the automata for all relative expressions that refer to the variable $\$v$ as context node are turned active. If we consider for example the source document and the XQuery transformation given in Fig. 1, we have the absolute XPath expression $XPa=/cont/country$, the results of which are bound to the variable $\$country$ and the relative XPath expression $XPr=\$country/city$. So whenever a result of XPa is found (e.g. in line (2) of the source document), the automaton for XPr is started, and from that moment on, the XML events are not only passed to the automaton of XPa , but also to the automaton of XPr . Only when the end of the result is reached, i.e., the end-tag of the root of the sub-tree that is the result of the query XPa is read (e.g. in line (5) of the source document), XPr is stopped, and it does not receive any further XML events.

In order to avoid any unnecessary parsing and decompression, the automata send feedback to the source, which navigation events are currently relevant, i.e., they report the set of labels of outgoing transitions of all currently active states. If we consider e.g. the absolute XPath expression $/cont/country$, and if the current node CN of any given source document is at level 2 (i.e., CN is a grand-child of the document root), navigating down to any level below via the first-child axis cannot lead to a result. Therefore, the whole sub-tree having the parent CN can be skipped and does not need to be decompressed.

This evaluation of absolute and relative XPath expressions describes a generic optimization for XQuery evaluation that can be applied to all XML representations supporting the basic operations, but this generic optimization need not be applied. Note that whenever an XML representation supports a more efficient XPath evaluation for a given XML document, this evaluation can be used instead of the generic part.

Navigation in Uncompressed XML. In order to navigate via the binary axes within an uncompressed XML document, we have to consider pairs of tags. Reading a start-tag following another start-tag corresponds to a first-child event. Reading a start-tag following an end-tag corresponds to a next-sibling event. Reading an end-tag following another end-tag corresponds to an end-of-parent event. Reading an end-tag following a start-tag does not generate an event at all. Finally, if the current tag is a start-tag with label 'a', this corresponds to a self::a event.

To skip a sub-tree means 'ignoring' its tags, i.e. skipping as many tags, until the number of the skipped start-tags is equal to the number of skipped end-tags.

Navigation in Succinct Compression. The Succinct Compression [6] stores the tree structure of the XML document in a bit stream that contains a '1'-bit for each start-tag and a '0'-bit for each end-tag within the document. The labels of the document tree are stored in so called inverted element lists, that contain for each label L the list of positions of '1'-bits within the bit stream that correspond to an element with label L.

For example, Fig. 3 shows the bit stream BS and the inverted element lists IES of the source document of Fig. 1(a) and the bit stream BT and the inverted element lists IET of the target document of Fig. 1(b). The data structure P is the index of positions within BS and BT to which the entries in the inverted element lists refer. The inverted element list with label "=-T" represents positions of text values within the bit stream, and each text node is represented as a bit sequence '10' in the bit stream.

BS:	1	1	1	1	0	0	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	0
P:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
BT:	1	1	1	1	0	0	1	1	0	0	0	1	1	1	0	0	0	0	1	1	0	0	0	0
IES:	continent: 0												IET:											
	country: 1, 17												mainland: 0											
	city: 2, 6, 18												nation: 1, 11											
	river: 11												city: 2, 6, 12											
	name: 12												=T: 3, 7, 13											
=T:	3, 7, 13, 19																							

Fig. 3. Bit stream BS and inverted element lists IES of the source document given in Fig. 1(a), and bit stream BT and inverted element lists IET of the target document given in Fig. 1(b)

In order to navigate within the Succinct compression, we have to consider two consecutive positions within the bit stream. Reading two ‘1’-bits corresponds to a first-child event. Reading a ‘1’-bit following a ‘0’-bit corresponds to a next-sibling event. Reading two ‘0’-bits corresponds to an end-of-parent event. If a self event is currently relevant, we have to search the current index position within the inverted element list in order to determine the label of the current node.

To skip a sub-tree means to proceed from the current position in the bit stream, until we have read as many ‘1’-bits as ‘0’-bits.

2.3 Copying Compressed Sub-Trees

Generic Copying of Sub-Trees. The generic approach to copy a sub-tree navigates through the sub-tree in the source document via the binary axes and inserts node by node into the target document via the operations `insertAsFirstChild` or `insertAsNextSibling`. As this corresponds to a decompression of the sub-tree to be copied, it is more efficient to provide XML-representation specific implementations for copying a sub-tree.

As the copying of sub-trees is only started after the XPath evaluation has stopped, all root nodes of the sub-trees to be copied are known in advance. Therefore, the whole copy-process can be performed within a (second) single pass through the source and through the target document, assuming that we can use a cache that allows us to store data that is read from the source document and will be appended to the target document at a later point in time.

Copying Sub-Trees in Uncompressed XML. For each sub-tree to be copied, we store the character index of the opening bracket ‘<’ of the start-tag during the XPath evaluation. In order to copy the sub-tree, the characters are copied and occurrences of start- and end-tags are counted until as many start-tags as end-tags have been copied. As target documents are generated in document order, the copied sub-tree is inserted ‘as a whole’ to the end of the target document.

Copying Sub-Trees in Succinct Compression. For each sub-tree to be copied, we store the position *n* of the ‘1’-bit that represents the start-tag of the sub-tree’s root within the bit stream. In the bit stream, we copy as many bits, until the number of ‘1’-bits that have been copied are equal to the number of ‘0’-bits that have been copied. Furthermore, the elements found at these positions in the inverted element lists have to be copied and their positions in the target bit stream have to be computed as described below.

In order to illustrate the process of copying a sub-tree from the source Succinct representation into the target Succinct representation, we assume that we want to copy the sub-tree BS' of the source document that is represented by the positions 18 to 21 of the bit stream BS given in Fig. 3. The affected bits and inverted element list entries are emphasized by bold letters. Furthermore, we assume that we want to append them to the target document that up to now consists of the positions 0 to 11 of the bit stream BT of the target document of Fig. 3, i.e., we want to insert the copied sub-tree at position $k=12$.

Having copied the bits from position n (i.e. 18) to position m (i.e. 21) of BS to BT , we run through the inverted element list and copy all positions x with $n \leq x \leq m$ (i.e. position 18 of the list “city” and position 19 of the list “=T”). As we only append new nodes to the current end of the target document, we do not have to re-compute index positions within the inverted element lists of nodes that have been previously added to the target document. In order to insert the copied sub-tree at position k (i.e. 12), we add $k-n$ to each copied position (i.e., we add $12-18=-6$ to the positions 18 and 19 and yield the new positions 12 and 13). Then, we can insert the list of positions of the copied sub-tree into the list of positions of the target document for each label (i.e., we add positions 12 to the list “city” of BT and position 13 to the list “=T” of BT).

Even if we have to copy more than one sub-tree, we can copy the relevant parts of the bit stream and the inverted element lists within a single pass. We first sort the positions of the sub-trees to be copied in ascending order, process the bit stream and copy all relevant bits, and determine the start and end positions of the copied sub-trees. Then, we can run a single pass through the inverted element lists in which we adapt the positions and insert the positions that were copied.

3 Evaluation of Our Prototype Implementation

To evaluate the performance of our Java prototype implementation, we conducted several tests. We ran the tests on an Intel Core i5 450M with 2.4 GHz and 4 GB memory using Linux and JRE 1.6.0. The tests compare our prototype implementation to version 1.4.0 of the XQuery evaluator Zorba (<http://www.zorba-xquery.com/>), started by `zorba -f -q queryFile -o outputFile`.

For each test, we compared five methods of transformation. Three methods use our prototype implementation: (1) transforming uncompressed data, (2) indirectly transforming data compressed via Succinct, i.e., first decompression, then uncompressed transformation, and, finally, compression of the transformation’s result and (3) direct transformation of data compressed via Succinct. Two methods use Zorba: (4) transforming uncompressed data and (5) indirectly transforming data compressed via Succinct as explained above. With Zorba, it is not possible to transform compressed data directly.

In the first test, we use an XML source file of variable size consisting of the root node `<Country>` that itself consists of a single `<Name>` node and of N `<City>` nodes. Based on that XML source file, we evaluate an XQuery that selects the root node `<Country>`. Afterwards, using the root node as context node, it evaluates the two relative XPath expressions `$country/Name` and `$country/City`. For each `<City>` node, the selected nodes are copied to the transformation’s target document.

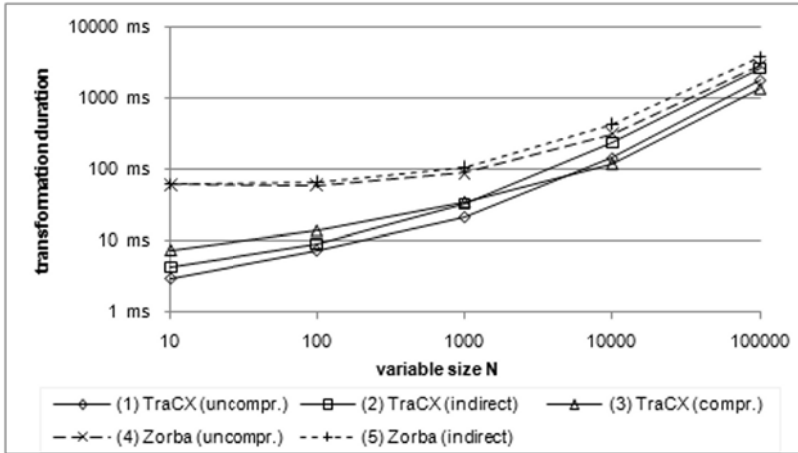


Fig. 4. Duration for transformation of Test 1 (logarithmically scaled)

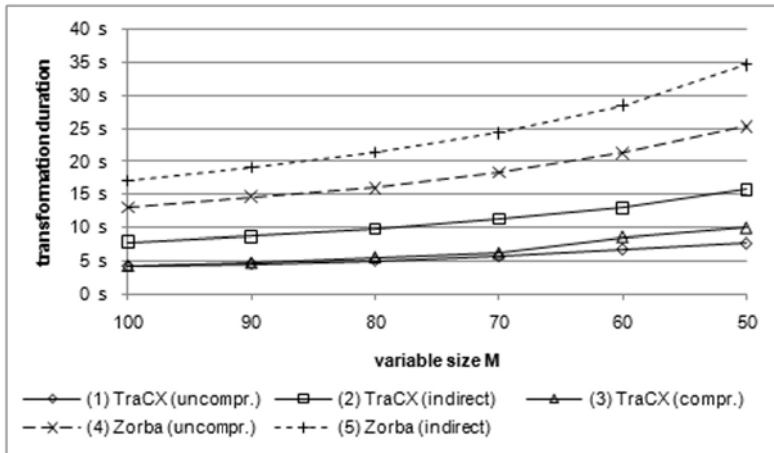


Fig. 5. Duration for transformation of Test 2

Fig. 4 shows that our prototype is able to perform the transformation faster than Zorba. Furthermore, for large source documents transforming compressed data is not only faster than the indirect way via decompression, transformation and compression but even faster than only transforming uncompressed data.

The second test uses an XML source file which consists of N `<CityAt>` nodes below the root node `<Country>`. The source file also contains a `<MinDistance>` node before each M 'th `<CityAt>` node. The XQuery evaluated for that source file selects all `<MinDistance>` nodes and uses each of these nodes as a context node for the relative XPath expression `$dis/following-sibling::CityAt`. Thus, for each `<MinDistance>` all cities are selected that are more distant. Again, the selected nodes are copied to the transformation's target document. For a given N , reducing M leads to an increasing

number of $\langle \text{MinDistance} \rangle$ nodes and, thus, an increasing number of context nodes for the relative XPath expression.

Fig. 5 illustrates that for $N=10,000$ our prototype implementation transforms uncompressed and compressed data faster than Zorba. In contrast to Zorba, our prototype implementation appears to benefit from the automata based evaluation of XPath expressions for compressed and uncompressed documents, resulting in a smooth increase of transformation duration for an increasing number of context elements for relative XPath expressions. Our tests for $N=1,000$ returned similar results.

The third test uses XML files of variable size generated by XMark [6] as source documents. The transformations reflect the following XPathMark [7] queries

- A1=/site/closed_auctions/closed_auction/annotation/description/text/keyword
- A2 =//closed_auction//keyword
- A3 =/site/closed_auctions/closed_auction//keyword

For each query $Q \in \{A1, A2, A3\}$ the result of Q is embedded into an XQuery transformation result that returns a $\langle \text{results} \rangle$ node as its root that contains a $\langle \text{result} \rangle$ node with R as its content for each query result R of Q .

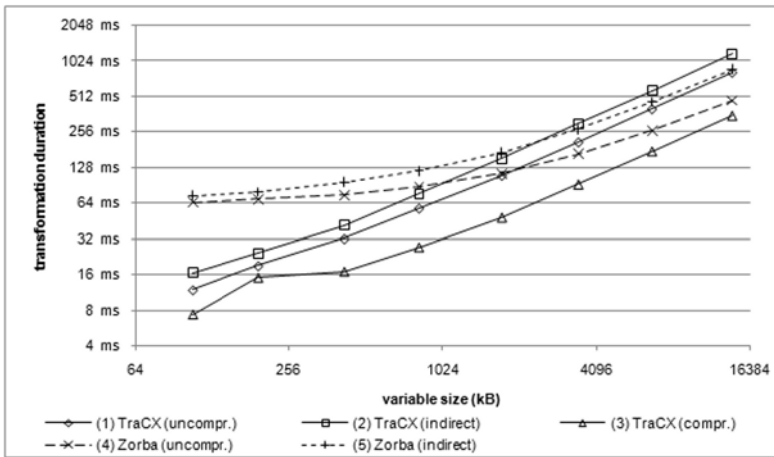


Fig. 6. Duration for transformation of Test 3, average of queries A1, A2, and A3 (logarithmically scaled)

For all the three queries of XPathMark, our tests yield very similar results, the average of which is shown in Fig. 6. While our prototype implementation transforms small uncompressed documents faster, Zorba is faster for large uncompressed documents. However, if compressed documents are given as input and required as output, our prototype implementation is always faster than Zorba. Furthermore, in contrast to Zorba, our prototype implementation reveals a linear growth of transformation duration.

4 Related Works

Most works that discuss the transformation of XML documents via XQuery focus on an efficient evaluation that can handle uncompressed XML only. A first example is [8][8] that discusses the unnesting of FLWOR expressions, while [9] introduces an algebra for XQuery and shows, how unnesting can be performed within this algebra. Further algebras for XQuery were introduced by [10] and [11]. The work of Fernández and Siméon was realized in an XQuery processor called Galax [12]. Within Galax, the internal data model is implemented for three different kinds of XML representations, but none of these is a compressed XML representation.

The evaluation of XQuery instructions on top of compressed XML data is performed by the approaches XQueC [2] and Bisimulation [3]. A document that is compressed via Bisimulation can be transformed with the help of XQuery into a compressed target document. In this approach, the XQuery language is restricted – similar as in our approach – to for, let and return expressions. In contrast to our approach, Bisimulation [3] supports XPath expressions consisting of child axis steps only, whereas our approach supports all forward axes except the following axis.

XQueC [2] proposes an XML representation that is optimized for an efficient transformation via XQuery. The structure compression as well as the data compression is chosen in such a way that path queries can be evaluated efficiently on it, but in return, the compression ratio reached by XQueC [2] is not as strong as by other compressors. The approaches being used in XQueC [2] appear to be non-applicable to other compression techniques.

Our approach contains parts that are generic, while other parts are specific for the compressed XML representation being used. Whenever another compressed XML representation supports the specific parts, our approach to transforming compressed XML representations can be applied to the other compressed XML representation as well.

To summarize, to the best of our knowledge, our approach is the only approach that provides a generic XQuery transformation of compressed XML data which is applicable to all compressed XML data formats that support the basic navigation steps first-child, next-sibling, end-of-parent, self::label, and that support inserting nodes as first-child or as next-sibling, and that optionally support a copy operation for compressed data.

5 Summary and Conclusions

Whenever the data volume of XML files to be exchanged or stored and to be transformed is a bottleneck of an application, transforming compressed XML is a promising alternative to decompressing compressed data back to XML prior to transforming it, and eventually recompressing the transformed XML data afterwards.

In this paper, we have presented a generic XML transformation approach that allows evaluating queries belonging to a subset of XQuery on all the compressed or uncompressed XML representations that support at least a small set of basic operations like the navigation via the binary XML axes first-child, next-sibling and end-of-parent, and like the insertion of nodes.

Our tests have shown that our XQuery processor is very efficient, such that it is not only faster to transform the compressed document directly than to use the indirection via decompression, XQuery transformation and recompression, but furthermore, that our XQuery processor reaches an evaluation speed on compressed XML that is similar to and sometimes even outperforms the speed of other XQuery processors on uncompressed XML.

As our approach is not limited to XQuery transformations of the Succinct XML representation [4], we consider it an interesting extension to integrate our approach to XQuery evaluation on compressed XML formats with other queryable and updateable XML compressors and to compare these XML compressors with plain XML regarding the XQuery transformation speed.

References

1. Scott Boag, D.: XQuery 1.0: An XML Query Language (2007)
2. Arion, A., Bonifati, A., Manolescu, I., Pugliese, A.: XQueC: A query-conscious compressed XML database. *ACM Trans. Internet Techn.* 7(2) (2007)
3. Buneman, P., Choi, B., Fan, W., Hutchison, R., Mann, R., Viglas, S.: Vectorizing and Querying Large XML Repositories. In: *ICDE 2005*, Tokyo, Japan, pp. 261–272 (2005)
4. Böttcher, S., Hartel, R., Heinzemann, C.: Compressing XML data streams with DAG+BSBC. In: Cordeiro, J., Hammoudi, S., Filipe, J. (eds.) *WEBIST 2008*. LNBI, vol. 18, pp. 65–79. Springer, Heidelberg (2009)
5. Böttcher, S., Steinmetz, R.: Evaluating XPath Queries on XML Data Streams. In: Cooper, R., Kennedy, J. (eds.) *BNCOD 2007*. LNCS, vol. 4587, pp. 101–113. Springer, Heidelberg (2007)
6. Schmidt, A., Waas, F., Kersten, M., Carey, M., Manolescu, I., Busse, R.: XMark: A Benchmark for XML Data Management. In: *VLDB 2002*, Hong Kong, China, pp. 974–985 (2002)
7. Franceschet, M.: XPathMark: An XPath Benchmark for the XMark Generated Data. In: Bressan, S., Ceri, S., Hunt, E., Ives, Z., Bellahsene, Z., Rys, M., Unland, R. (eds.) *XXSym 2005*. LNCS, vol. 3671, pp. 129–143. Springer, Heidelberg (2005)
8. Deutsch, A., Papanikolaou, Y., Xu, Y.: The NEXT Logical Framework for XQuery. In: *VLDB*, Toronto, Canada, pp. 168–179 (2004)
9. Re, C., Siméon, J., Fernández, M.: A Complete and Efficient Algebraic Compiler for XQuery. In: *ICDE 2006*, Atlanta, GA, USA, p.14 (2006)
10. Zhang, X., Pielech, B., Rundensteiner, E.: Honey, I shrunk the XQuery!: an XML algebra optimization approach. In: *WIDM 2002*, LcLean, Virginia, USA, pp. 15–22 (2002)
11. Fernández, M., Michiels, P., Siméon, J., Stark, M.: XQuery Streaming à la Carte. In: *ICDE 2007*, Istanbul, Turkey, pp. 256–265 (2007)
12. Fernández, M., Siméon, J., Choi, B., Marian, A., Sur, G.: Implementing Xquery 1.0: The Galax Experience. In: *VLDB*, pp. 1077–1080 (2003)

Satisfiability of Simple XPath Fragments under Fixed DTDs

Nobutaka Suzuki

University of Tsukuba

1-2, Kasuga, Tsukuba Ibaraki 305-8550, Japan
nsuzuki@slis.tsukuba.ac.jp

Abstract. The XPath satisfiability problem is to decide, for an XPath expression q and a DTD D , if there exists an XML document t valid against D such that the answer of q on t is nonempty. It is shown that the satisfiability problem is intractable for many XPath fragments. In this paper, we focus on fixed DTDs and consider the problem under fixed DTDs. We first show that, for a very restricted XPath fragment, the satisfiability problem is NP-complete under fixed DTDs. Then we show several XPath fragments for which satisfiability is in PTIME under fixed DTDs.

1 Introduction

XPath is a widely accepted query language for XML. For an XPath expression q and a DTD D , q is *satisfiable* under D if there exists an XML document t valid against D such that the answer of q on t is nonempty. Clearly, evaluating an unsatisfiable XPath expression is meaningless since such an expression can always be replaced by an empty set without evaluating it. However, it is shown that the satisfiability problem is intractable for a large number of XPath fragments [14]. Therefore, it is important to find XPath fragments for which the satisfiability problem is solvable efficiently. In this paper, we focus on fixed DTDs and consider XPath fragments for which satisfiability can be determined efficiently under fixed DTDs.

Let us show a simple example of an unsatisfiable XPath expression. Let us consider the following DTD.

```
<!ELEMENT students      (undergraduate|graduate)+>
<!ELEMENT undergraduate (name,email)>
<!ELEMENT graduate      (name,email,supervisor?)>
<!ELEMENT name          (#PCDATA)>
<!ELEMENT email         (#PCDATA)>
<!ELEMENT supervisor    (#PCDATA)>
```

Let q be an XPath expression `//supervisor/parent::undergraduate/name`, which would return the names of undergraduate students that have a supervisor. However, it is easy to see that q is unsatisfiable since an `undergraduate`

element cannot have any **supervisor** element as a child. Clearly, we should detect unsatisfiable XPath expressions prior to evaluating them. Actually, since the sizes of a DTD D and a query q are much smaller than that of a database, whether q is satisfiable under D can be checked much faster than evaluating q on the database. Therefore, by checking the satisfiability of q before evaluating q , we can avoid evaluating q and thus save substantial time whenever q is unsatisfiable.

There have been a number of studies on the XPath satisfiability problem, and many of the studies are considered under “unfixed” DTDs so far, where term unfixed (fixed) means that the complexity of the satisfiability problem is measured by the sizes of a DTD and an XPath expression (resp., only the size of an XPath expression). On the other hand, it also makes sense that the satisfiability problem is considered under *fixed* DTDs. Actually, a schema is usually much smaller than its underlying database, and a query issued to a database varies almost every time but the schema of the database is stable and hardly changes, i.e., schema is “fixed”. Thus, in this paper we consider XPath fragments for which satisfiability is in PTIME under fixed DTDs.

In this paper, we use simple XPath fragments using child (\downarrow), descendant-or-self (\downarrow^*), parent (\uparrow), ancestor-or-self (\uparrow^*), following-sibling (\rightarrow^+), and preceding-sibling (\leftarrow^+) axes under two restrictions; (i) only a label can be specified as a node test and (ii) operators such as qualifier ($[]$) and path union (\cup) are not allowed. This XPath fragment is rather simple but we believe that the fragment covers a most important part of the full XPath since the fragment also covers descendant, ancestor, following, and preceding axes (e.g., following axis can be simulated by \uparrow^* , \rightarrow^+ , and \downarrow^* axes) as well as the six axes mentioned above. Moreover, our fragment also supports a qualifier containing only child axes, since an XPath expression having such a qualifier can be converted into an equivalent expression without qualifier, e.g., $/a/b[c/d]/e$ is equivalent to $/a/b/c/d/\uparrow::c/\uparrow::b/e$.

In this paper, we first show that satisfiability for $XP^{\{\downarrow,\uparrow\}}$ is NP-complete under fixed DTDs, where $XP^{\{\downarrow,\uparrow\}}$ denotes the XPath fragment using only \downarrow and \uparrow axes. We next show that satisfiability for $XP^{\{\downarrow,\downarrow^*,\rightarrow^+,\leftarrow^+\}}$ is in PTIME under fixed DTDs. This contrasts with the NP-hardness of satisfiability for $XP^{\{\downarrow,\rightarrow^+,\leftarrow^+\}}$ under unfixed DTDs [10]. Finally, we show sufficient conditions under which satisfiability for $XP^{\{\downarrow,\downarrow^*,\uparrow,\uparrow^*,\rightarrow^+,\leftarrow^+\}}$ is in PTIME under fixed DTDs.

Benedikt, Fan, and Geerts extensively study the XPath satisfiability problem [14], in which many XPath fragments remain intractable even under fixed DTDs. In this paper, we consider XPath fragments that are not considered in their studies. There also have been many studies on the XPath satisfiability problem without DTDs or under unfixed DTDs. Hidders considered the XPath satisfiability problem without DTDs [5]. Figueira investigated satisfiability for $XP^{\{\downarrow,\downarrow^*,=\}}$ without DTDs and showed that the problem is EXPTIME-complete [3]. Lakshmanan et al. considered the satisfiability problem for tree pattern queries (i.e., $XP^{\{\downarrow,\downarrow^*,[]\}}$) without DTDs and under unfixed DTDs [7]. Their tree pattern queries and the XPath fragments in this paper are incomparable in the sense that their data model is based on an unordered tree model

but ours is based on an ordered tree model (sibling axes must be handled differently). Several studies focused on restricted (unfixed) DTDs and XPath fragments for which the problem is solvable efficiently. Montazerian et al. proposed two subclasses of DTDs called duplicate-free DTDs and covering DTDs, and showed that satisfiability for some subfragments of $XP^{\{\downarrow, \downarrow^*, \uparrow, \cup, *\}}$ can be solved in PTIME under these DTDs [8]. Ishihara et al. proposed a subclasses of covering DTDs and investigated the tractability of XPath satisfiability under the subclasses [6]. Suzuki et al. considered some XPath fragments for which satisfiability is in PTIME under unfixed duplicate-free DTDs [10]. Their studies focus on restricted unfixed DTDs, while this paper assumes that DTDs are fixed but do not issues such restrictions.

The rest of this paper is organized as follows. Section 2 gives some preliminaries. Section 3 shows the NP-completeness of satisfiability for $XP^{\{\downarrow, \uparrow\}}$ under fixed DTDs. Section 4 presents a polynomial-time algorithm that solves satisfiability for $XP^{\{\downarrow, \downarrow^*, \rightarrow^+, \leftarrow^+\}}$ under fixed DTDs. Section 5 shows a polynomial-time algorithm that solves satisfiability for $XP^{\{\downarrow, \downarrow^*, \uparrow, \uparrow^*, \rightarrow^+, \leftarrow^+\}}$ under fixed nonrecursive DTDs. Section 6 summarizes the paper.

2 Definitions

An XML document is modeled as a node-labeled ordered tree (attributes are omitted). Each node in a tree represents an element. A text node is omitted, in other words, we assume that each leaf node has a text node implicitly. For a node n in a tree, by $l(n)$ we mean the label (element name) of n . In what follows, we use the term tree when we mean node-labeled ordered tree unless otherwise stated.

Let Σ be a set of labels. A DTD is a tuple $D = (d, s)$, where d is a mapping from Σ to the set of regular expressions over Σ and $s \in \Sigma$ is the *start label*. For a label $a \in \Sigma$, $d(a)$ is the *content model* of a . For example, the DTD in Sect. 1 is denoted as follows.

$$\begin{aligned} d(\text{students}) &= (\text{undergraduate}|\text{graduate})^+ \\ d(\text{undergraduate}) &= \text{name, email} \\ d(\text{graduate}) &= \text{name email supervisor?} \\ d(\text{name}) &= \epsilon \\ d(\text{email}) &= \epsilon \\ d(\text{supervisor}) &= \epsilon \end{aligned}$$

A tree t is *valid* against D if (i) the root of t is labeled by s and (ii) for each node n in t $l(n_1) \cdots l(n_m) \in L(d(l(n)))$, where $n_1 \cdots n_m$ are the children of n and $L(d(l(n)))$ is the language of $d(l(n))$. D is *no-star* if for any content model $d(a)$ of D , $d(a)$ contains no ‘*’ operator. For labels a, b in D , we say that b is *reachable* from a if b occurs in $d(a)$ or there is a label c such that c is reachable

from a and that b occurs in $d(c)$. D is *recursive* if there are labels a, b in D such that a is reachable from b and b is reachable from a .

A *location step* is of the form $ax::lb$, where (i) ax is either \downarrow (the child axis), \downarrow^* (the descendant-or-self axis), \uparrow (the parent axis), \uparrow^* (the ancestor-or-self axis), \rightarrow^+ (the following-sibling axis), or \leftarrow^+ (the preceding-sibling axis), and (ii) lb is a label. A *query* is of the form $/ls_1/ls_2 \cdots /ls_n$, where ls_i is a location step. For example, query $/\downarrow^*::a/\rightarrow^+::b/c$ stands for

$$/\text{descendant-or-self}::a/\text{following-sibling}::b/\text{child}::c.$$

Let XP be the set of queries. We denote a fragment of XP by listing the axes supported by the fragment. For example, $XP^{\{\downarrow, \downarrow^*\}}$ denotes the set of queries using only child and descendant-or-self axes.

Let t be a tree and q be a query. We say that t *satisfies* q if the answer of q on t is nonempty. If there is a tree t such that t is valid against a DTD D and that t satisfies q , then q is *satisfiable* under D . For an XPath fragment XP^S , the *XPath satisfiability problem* for XP^S , denoted $\text{SAT}(XP^S)$, is to decide, for a DTD D and a query $q \in XP^S$, if q is satisfiable under D .

3 NP-Completeness

In this section, we consider the complexity of $\text{SAT}(XP^{\{\uparrow, \downarrow\}})$ under fixed DTDs. It is shown that $\text{SAT}(XP^{\{\uparrow, \downarrow\}})$ is NP-complete under fixed DTDs if a wildcard is allowed as a node test (Theorem 6.6 of Ref. [11]). It is also shown that $\text{SAT}(XP^{\{\uparrow, \downarrow\}})$ is NP-complete under unfixed DTDs assuming that only a label is allowed as a node test (Theorem 1 of Ref. [10]). In this section, we show a slightly more strong result; $\text{SAT}(XP^{\{\uparrow, \downarrow\}})$ is NP-complete under fixed DTDs even if only a label is allowed as a node test. This implies that the XPath satisfiability problem is still intractable under a very restricted setting.

Theorem 1. *$\text{SAT}(XP^{\{\uparrow, \downarrow\}})$ is NP-complete under fixed DTDs, even if only a label is allowed as a node test.*

Proof. For a query q , a DTD D , and a tree t valid against D , it is clear that whether the answer of q on t is nonempty can be checked in polynomial time. Thus $\text{SAT}(XP^{\{\uparrow, \downarrow\}})$ is in NP.

To show the NP-hardness of the problem, we reduce 3SAT to this problem, similarly to Theorem 6.6 of Ref. [11]. However, our reduction is much more complicated since no wildcard node test is allowed. Let $\phi = C_1 \wedge \cdots \wedge C_n$ be an instance of 3SAT, where C_1, \dots, C_n are clauses. Let x_1, \dots, x_m be the variables in ϕ . Without loss of generality, we assume that for any variable x_i , positive literal x_i and negative literal $\neg x_i$ do not appear in the same clause. From this instance, we construct an instance of the XPath satisfiability problem under fixed DTDs, as follows.

First, DTD $D = (d, r)$ is defined as follows.

$$\begin{aligned} d(r) &= x, \\ d(x) &= (E'_x|x)(x|B_c|E_x), \\ d(B_c) &= c c, \\ d(c) &= (T_c|F_c)(c|E_c), \\ d(E_x) &= d(E'_x) = d(E_c) = d(T_c) = d(F_c) = \epsilon. \end{aligned}$$

Note that D is independent of ϕ since D is fixed. In contrast, D depends on ϕ in the proof of Theorem 1 in Ref. [10].

Second, query q is defined as follows.

$$q = \overbrace{/r/x/ \cdots /x/E_x}^m \tag{1}$$

$$/ \uparrow:: x/P_m/ \uparrow:: x/P_{m-1}/ \uparrow:: x/\cdots/ \uparrow:: x/P_1/ \uparrow:: x/ \uparrow:: r \tag{2}$$

$$/Q_1/Q_2/\cdots/Q_n, \tag{3}$$

where P_m, P_{m-1}, \dots, P_1 in (2) and Q_1, Q_2, \dots, Q_n in (3) are subqueries defined later. In short, according to ϕ subqueries (1) and (2) plot a tree t shown in Fig. 1, and then subquery (3) checks whether ϕ is satisfiable over t . Each subquery P_i plots the “subtree” P_i in t , as shown in Fig. 1. Formally, P_i is defined as follows ($1 \leq i \leq m$).

$$P_i = \overbrace{/x/E'_x/ \uparrow:: x/\cdots/x/E'_x/ \uparrow:: x/B_c}^{m-i+1} \tag{4}$$

$$/c/T_{i,1}/ \uparrow:: c/c/T_{i,2}/ \uparrow:: c/\cdots/c/T_{i,n-1}/ \uparrow:: c/c/T_{i,n}/ \uparrow:: c \tag{5}$$

$$\overbrace{/ \uparrow:: c/\cdots/ \uparrow:: c/B_c}^{n+1} \tag{6}$$

$$/c/F_{i,1}/ \uparrow:: c/c/F_{i,2}/ \uparrow:: c/\cdots/c/F_{i,n-1}/ \uparrow:: c/c/F_{i,n}/ \uparrow:: c \tag{7}$$

$$\overbrace{/ \uparrow:: c/\cdots/ \uparrow:: c/B_c}^{n+1} \tag{8}$$

$$\overbrace{/c/\cdots/c/T_c/ \uparrow:: c/E_c/ \uparrow:: c/\cdots/ \uparrow:: c/ \uparrow:: B_c}^{n+1} \tag{9}$$

$$\overbrace{/c/\cdots/c/F_c/ \uparrow:: c/E_c/ \uparrow:: c/\cdots/ \uparrow:: c/ \uparrow:: B_c}^{n+1} \tag{10}$$

$$\overbrace{/ \uparrow:: x/\cdots/ \uparrow:: x}^{m-i+1} \tag{11}$$

Starting from the x -node at position x_i in t (Fig. 1), the context node goes down to B_c by (4). Then subquery (5) sets the value of $T_{i,j}$ for $1 \leq j \leq n$ and subquery (7) sets those of $F_{i,j}$ for $1 \leq j \leq n$, where

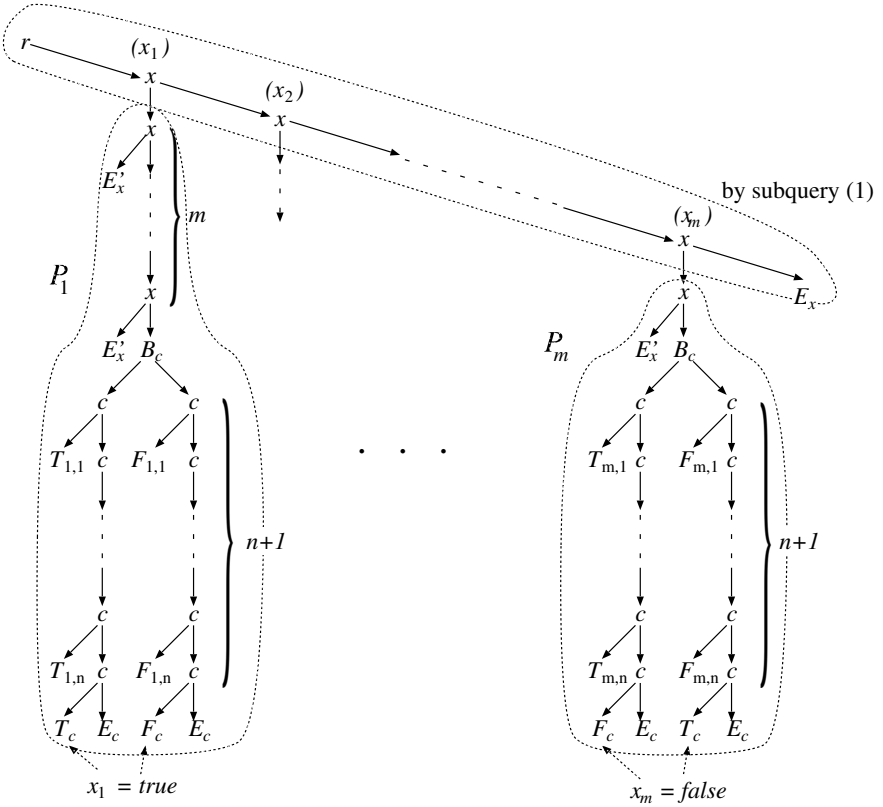


Fig. 1. Tree t plotted by subqueries (1) and (2)

$$T_{i,j} = \begin{cases} T_c & \text{if positive literal } x_i \text{ occurs in } C_j, \\ F_c & \text{otherwise,} \end{cases}$$

$$F_{i,j} = \begin{cases} T_c & \text{if negative literal } \neg x_i \text{ occurs in } C_j, \\ F_c & \text{otherwise.} \end{cases}$$

Thus, if $T_{i,j} = T_c$, then clause C_j in ϕ becomes true by setting $x_i = true$, and if $F_{i,j} = T_c$, then C_j becomes true by setting $x_i = false$. The value of x_i is set by subqueries (9) and (10), as follows. In t , the value of x_i is identified by the labels of two bottom leaf nodes of subtree P_i , where one is labeled by T_c and the other is labeled by F_c . For example, $x_1 = true$ and $x_m = false$ in Fig. 1. More precisely, let t_{T_i} and t_{F_i} be the subtrees plotted by subqueries (5) and (7), respectively. If the left bottom leaf node of t_{T_i} (t_{F_i}) is labeled by T_c , then x_i is considered to be *true* (resp., *false*).

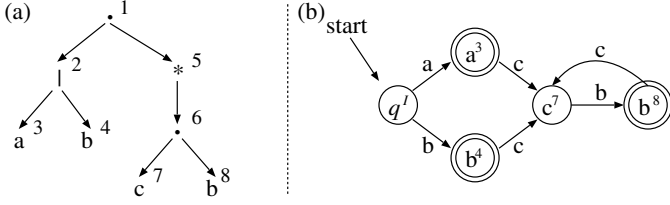


Fig. 2. (a) tree representation of $r = (a|b)(cb)^*$ and (b) the Glushkov automaton of r

Finally, let us define subquery Q_i used in (3) ($1 \leq i \leq n$).

$$Q_i = \underbrace{/x/\cdots/x/B_c/c/\cdots/c/T_c/}_{m+1} \underbrace{\uparrow::c/T_c/}_{i} \underbrace{\uparrow::c/c/\cdots/c/T_c}_{n-i+1} \quad (12)$$

$$\underbrace{/\uparrow::c/\cdots/\uparrow::c/\uparrow::B_c/}_{n+1} \underbrace{\uparrow::x/\cdots/\uparrow::x/\uparrow::r}_{m+1} \quad (13)$$

Starting from the root r , subquery (12) checks if clause C_i is true. Then the context node goes back to the root r by subquery (13).

Now it is easy show that ϕ is satisfiable iff q is satisfiable under D .

4 Satisfiability Problem without Upward Axis

As shown in the previous section, the satisfiability problem is intractable under fixed DTDs if upward axes are allowed. In this section, we consider the satisfiability problem without upward axes, i.e., $\text{SAT}(\text{XP}^{\{\downarrow, \downarrow^*, \rightarrow^+, \leftarrow^+\}})$ under fixed DTDs. It is shown that $\text{SAT}(\text{XP}^{\{\downarrow, \rightarrow^+, \leftarrow^+\}})$ is NP-complete under unfixed DTDs [10]. On the other hand, we show in this section that $\text{SAT}(\text{XP}^{\{\downarrow, \downarrow^*, \rightarrow^+, \leftarrow^+\}})$ is in PTIME under fixed DTDs.

In the following, we first show Glushkov automaton, which is used in the algorithms proposed in this and the next sections. Then we show a polynomial-time algorithm for solving $\text{SAT}(\text{XP}^{\{\downarrow, \downarrow^*, \rightarrow^+, \leftarrow^+\}})$ under fixed DTDs.

4.1 Glushkov Automaton

Let r be a regular expression. We associate each label occurring in r with a unique number to identify the label; we use the tree representation of r (Fig. 2(a)), and assume that each node is numbered in prefix order, called *position*. By $r^\#$ we mean the *superscripted* regular expression of r obtained by superscripting each label occurring in r by its corresponding position. By $\text{sym}(r^\#)$ we mean the set of superscripted labels occurring in $r^\#$. For example, if $r = (a|b)(cb)^*$, then $r^\# = (a^3|b^4)(c^7b^8)^*$ and $\text{sym}(r^\#) = \{a^3, b^4, c^7, b^8\}$. Let a^i be a superscripted label of a . By $(a^i)^\natural$ we mean the label resulting from a^i by dropping the superscript of a^i , that is, $(a^i)^\natural = a$.

The *Glushkov automaton* of a regular expression r is a 5-tuple $G_r = (Q, \Sigma_r, \delta, q^I, F)$, where $Q = \text{sym}(r^\#) \cup \{q^I\}$ is a set of *states*, Σ_r is the set of labels occurring in r , δ is a *transition function*, $q^I \notin \text{sym}(r^\#)$ is the *start state* of G_r , and F is a set of *final states*. Because of space limitation, the definitions of δ and F are skipped (details can be found in [2,9]). Let us show an example instead. Figure 2(b) shows the Glushkov automaton $G_r = (Q, \Sigma_r, \delta, q^I, F)$ of $r = (a|b)(cb)^*$, where $Q = \{q^I, a^3, b^4, c^7, b^8\}$, $\Sigma_r = \{a, b, c\}$, $F = \{a^3, b^4, b^8\}$, and δ is a transition function defined as follows.

$$\begin{aligned}\delta(q^I, a) &= \{a^3\}, \\ \delta(q^I, b) &= \{b^4\}, \\ \delta(a^3, c) &= \delta(b^4, c) = \delta(b^8, c) = \{c^7\}, \\ \delta(c^7, b) &= \{b^8\}.\end{aligned}$$

For any regular expression r , it holds that $L(r) = L(G_r)$ and that there is a one-to-one correspondence between $\text{sym}(r^\#) \setminus \{q^I\}$ and the set of states of G_r . This property is essential to our algorithms shown below.

4.2 Polynomial-Time Algorithm

We show a polynomial-time algorithm for solving $\text{SAT}(\text{XP}^{\{\downarrow, \downarrow^*, \rightarrow^+, \leftarrow^+\}})$ under fixed DTDs. For a regular expression r , by $E(r)$ we mean the set of *extracted regular expressions* obtained by extracting each ‘|’ operator that is not under any ‘*’ operator. For example, if $r = (a|b)^*c(d|e|f)$, then $E(r) = \{(a|b)^*cd, (a|b)^*ce, (a|b)^*cf\}$.

Let us first show the outline of the algorithm. Let $q = /ax[1] :: lb[1]/\dots/ax[m] :: lb[m]$ be a query and $D = (d, s)$ be a fixed DTD. The algorithm uses the Glushkov automaton of the content model of each label in D . In short, for each $i = 1, \dots, m$, the algorithm computes the set of states of the Glushkov automata in D reachable from s via $/ax[1] :: lb[1]/\dots/ax[i] :: lb[i]$ under D , and returns “no” (i.e., q is unsatisfiable) if the set becomes empty. More concretely, the algorithm computes a set S_i for each $i = 1, \dots, m$, and returns “no” if $S_i = \emptyset$ for some i , returns “yes” otherwise. Here, S_i is a set of pairs (r, St) , where r is an extracted regular expression of the “current” content model and St is the set of states in G_r reachable from s via $/ax[1] :: lb[1]/\dots/ax[i] :: lb[i]$ under D .

Now let us show the “main” part of the algorithm. To obtain S_i , according to the current location step $ax[i] :: lb[i]$ we use four subroutines in lines 2 to 9 (defined later).

Input: A query $q = /ax[1] :: lb[1]/\dots/ax[m] :: lb[m]$ and a fixed DTD $D = (d, s)$.
Output: “yes” or “no”.

begin

1. **for** $i = 1$ **to** m **do**
2. **if** $ax[i] = \downarrow$ **then**
3. $S_i \leftarrow \text{do_child}(D, q, i)$;
4. **else if** $ax[i] = \downarrow^*$ **then**

```

5.    $S_i \leftarrow \text{do\_descendant-or-self}(D, q, i)$ ;
6.   else if  $ax[i] = \rightarrow^+$  then
7.      $S_i \leftarrow \text{do\_following-sibling}(D, q, i)$ ;
8.   else if  $ax[i] = \leftarrow^+$  then
9.      $S_i \leftarrow \text{do\_preceding-sibling}(D, q, i)$ ;
10.  end
11.  if  $S_i = \emptyset$  then
12.    return “no”;
13.  end
14. end
15. return “yes”;
end

```

Let us show the subroutines. We first show `do_descendant-or-self`. Since $ax[i] = \downarrow^*$, there may be more than one labels that can be both the descendant of $lb[i-1]$ and the parent of $lb[i]$. The set L of such labels is obtained in line 2. Then the union E of $E(d(l))$ of every $l \in L$ is obtained (line 3), and for each $r \in E$ a pair (r, St) is obtained and added to S (lines 4 to 9).

`do_descendant-or-self`(D, q, i)

```

begin
1.  $S \leftarrow \emptyset$ ;
2.  $L \leftarrow \{l \mid l = lb[i-1] \text{ or } l \text{ is reachable from } lb[i-1] \text{ in } D, d(l) \text{ contains } lb[i]\}$ ;
3.  $E \leftarrow \bigcup_{l \in L} E(d(l))$ ;
4. for each  $r \in E$  do
5.    $St \leftarrow \{a^i \mid a^i \text{ is a state in } G_r \text{ such that } (a^i)^\natural = lb[i]\}$ ;
6.   if  $St \neq \emptyset$  then
7.      $S \leftarrow S \cup \{(r, St)\}$ ;
8.   end
9. end
10. return  $S$ ;
end

```

`do_child` is defined similarly to `do_descendant-or-self`. The only difference is that in line 2 singleton $\{lb[i-1]\}$ is assigned to L instead of above.

We next show `do_following-sibling` (`do_preceding-sibling` is defined similarly). Since $ax[i] = \rightarrow^+$, for each $(r, St) \in S_{i-1}$ it suffices to find the states in G_r reachable from a state in St .

`do_following-sibling`(D, q, i)

```

begin
1.  $S \leftarrow \emptyset$ ;
2. for each  $(r, St) \in S_{i-1}$  do
3.    $St' \leftarrow \{a^i \mid (a^i)^\natural = lb[i], a^i \text{ is reachable from a state in } St \text{ in } G_r\}$ ;
4.   if  $St' \neq \emptyset$  then
5.      $S \leftarrow S \cup (r, St')$ ;
6.   end
7. end
8. return  $S$ ;
end

```

It is easy to show by induction on i that $(r, St) \in S_i$ iff there is a tree t valid against D such that t contains a node reachable from $/ax[1] :: lb[1]/\dots/ax[i] ::$

$lb[i]$. Moreover, since D is fixed, it is clear that the algorithm runs in $O(m)$ time, where m is the number of location steps in q . Thus we have the following.

Theorem 2. $SAT(XP^{\{\downarrow, \downarrow^*, \rightarrow^+, \leftarrow^+\}})$ is solvable in linear time under fixed DTDs.

5 Using Upward Axes under Fixed DTDs

We have shown that under fixed DTDs $SAT(XP^{\{\downarrow, \downarrow^*, \rightarrow^+, \leftarrow^+\}})$ is in PTIME but $SAT(XP^{\{\downarrow, \uparrow\}})$ is NP-complete. In this section, we show sufficient conditions under which $SAT(XP^{\{\downarrow, \downarrow^*, \uparrow, \uparrow^*, \rightarrow^+, \leftarrow^+\}})$ is in PTIME under fixed DTDs.

Suppose first that a DTD D is fixed, nonrecursive, and no-star. To check whether $q \in XP^{\{\downarrow, \downarrow^*, \uparrow, \uparrow^*, \rightarrow^+, \leftarrow^+\}}$ is satisfiable under D , it suffices to construct the set T of trees valid against D and check whether there is a tree $t \in T$ on which the result of q is nonempty. Since the size of T is fixed, $SAT(XP^{\{\downarrow, \downarrow^*, \uparrow, \uparrow^*, \rightarrow^+, \leftarrow^+\}})$ is clearly in PTIME under fixed, nonrecursive, no-star DTDs.

We next show a polynomial-time algorithm for deciding if a query $q \in XP^{\{\downarrow, \downarrow^*, \uparrow, \uparrow^*, \rightarrow^+, \leftarrow^+\}}$ is satisfiable under a fixed nonrecursive DTD D . In this case, there may be infinite trees valid against D since D may contain ‘*’ operators. Thus we have to construct a set T of valid trees carefully so that T is finite and contains valid trees enough to check the satisfiability of q under D .

Let $D = (d, s)$ be a fixed nonrecursive DTD and $q \in XP^{\{\downarrow, \downarrow^*, \uparrow, \uparrow^*, \rightarrow^+, \leftarrow^+\}}$ be a query. In short, our algorithm decides whether q is satisfiable under D , as follows.

1. For a label a and an extracted regular expression $r \in E(d(a))$, we use a set $T(a, r)$ of trees valid against DTD (d, a) such that $T(a, r)$ is finite but covers valid trees enough to check the satisfiability of q under (d, a) . For any $t \in T(a, r)$, each node n of t has a *type* denoted $type(n)$, where $type(n) \in E(d(l(n)))$ (in particular, $type(n) = r$ if n is the root of t), and the children of n match $type(n)$. $type(n)$ is used to handle \rightarrow^+ and \leftarrow^+ axes. We assume that every tree in $T(a, r)$ is unordered; the order among sibling nodes of a node n is managed by the Glushkov automaton of $type(n)$.
2. Let $T(s) = \bigcup_{r \in E(d(s))} T(s, r)$. If there is a tree $t \in T(s)$ on which the result of q is nonempty, then the algorithm returns “yes” (i.e., q is satisfiable), otherwise it returns “no”.

Example 1. Let $D = (d, s)$ be a DTD, where $d(s) = a(b|c)$, $d(b) = e|f$, $d(a) = d(c) = d(e) = d(f) = \epsilon$. We have $E(d(s)) = \{ab, ac\}$, thus $T(s) = T(s, ab) \cup T(s, ac)$. $T(s)$ consists of three trees (Fig. 3). In this case, $T(s)$ covers all the trees valid against D . As defined later, the label of each node is superscripted.

To define $T(a, r)$ formally, we need some definitions. If a superscripted label a^i is a descendant of a ‘*’ operator in the tree representation of $r^\#$, then we say that a^i is *starred*. By $sym^*(r^\#)$, we mean the set of superscripted starred labels in $r^\#$. For example, if $r^\# = a^3b^4(c^7|b^8)^*$, then $sym^*(r^\#) = \{c^7, b^8\}$. For a node n and trees t_1, \dots, t_m , by $n(t_1, \dots, t_m)$ we mean the tree rooted at n with m

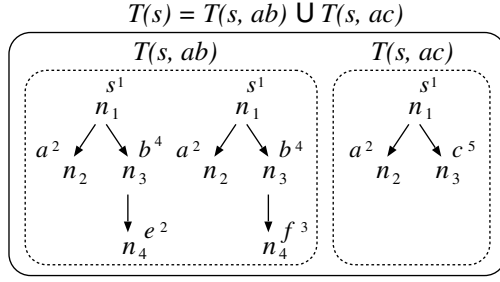


Fig. 3. An example of $T(s)$

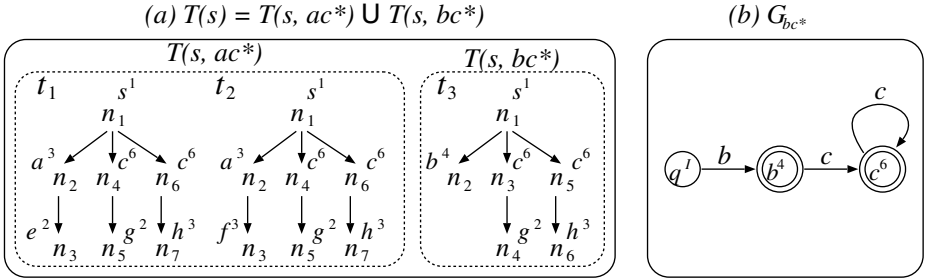


Fig. 4. (a) $T(s)$ and (b) Glushkov automaton G_{bc^*}

subtrees t_1, \dots, t_m . For a tree t , by $t(a^i)$ we mean the tree obtained from t by replacing the label of the root of t by a^i .

We now define $T(a, r)$ formally. For a label a and $r \in E(d(a))$, $T(a, r)$ is a set of trees defines as follows. We assume that every tree in $T(a, r)$ is unordered.

$$T(a, r) = \{n(F, F_{a^1}^*, \dots, F_{a^m}^*) \mid n \text{ is a node satisfying (A)},$$

$$F \text{ is a forest satisfying (B)}, F_{a^1}^*, \dots, F_{a^m}^* \text{ are forests satisfying (C)}\},$$

(A) $l(n) = a^1$ and $type(n) = r$.

(B) Let $sym(r^\#) \setminus sym^*(r^\#) = \{a^1, \dots, a^k\}$. Then $F = t_1(a^1), \dots, t_k(a^k)$, where $t_i \in T((a^i)^\natural, r_i)$ and $r_i \in E(d((a^i)^\natural))$ ($1 \leq i \leq k$).

(C) Let $sym^*(r^\#) = \{a^1, \dots, a^m\}$. Then $F_{a^i}^* = t_1(a^i), \dots, t_{k_i}(a^i)$ ($1 \leq i \leq m$), where $\{t_1, \dots, t_{k_i}\} = T((a^i)^\natural) = \bigcup_{r \in E(d((a^i)^\natural))} T((a^i)^\natural, r)$.

Example 2. Let $D = (d, s)$ be a DTD, where $d(s) = (a|b)c^*$, $d(a) = e|f$, $d(c) = g|h$, $d(b) = d(g) = d(h) = \epsilon$. $T(s) = T(s, ac^*) \cup T(s, bc^*)$ is shown in Fig. 4(a).

We need one more definition. Let S be a set of nodes in a tree t . The s -partition of S is a partition of S such that for any nodes $n, n' \in S$, n and n' belong to the same subset iff n and n' are siblings in t . Thus, for the s -partition P_1, \dots, P_k of S , every node in P_i has the same parent node, say n_p , for any $1 \leq i \leq k$, and n_p is called the *parent* of P_i .

We now show an algorithm for deciding if a query $q = /ax[1] :: lb[1]/\dots/ax[m] :: lb[m]$ in $XP^{\{\downarrow, \downarrow^*, \uparrow, \uparrow^*, \rightarrow^+, \leftarrow^+\}}$ is satisfiable under a fixed nonrecursive DTD D . The algorithm computes for each $t \in T(s)$ and for each $i = 1 \dots m$, set S_i of nodes reachable from the root of t via $/ax[1] :: lb[1]/\dots/ax[i] :: lb[i]$. The algorithm returns “yes” if $S_m \neq \emptyset$ for some $t \in T(s)$, returns “no” otherwise. If $ax[i] \in \{\downarrow, \downarrow^*, \uparrow, \uparrow^*\}$, then S_i is easily obtained from S_{i-1} and $ax[i] :: lb[i]$ (lines 6 to 13). Let us consider the case where $ax[i] = \rightarrow^+$. The algorithm (i) computes the s-partition P_1, \dots, P_k of S_{i-1} (line 15), (ii) finds $S_{i,j}$ for each P_j (lines 16 to 21), where $S_{i,j}$ is the set of nodes in t reachable from a node in P_j via location step $ax[i] :: lb[i]$, and (iii) computes $S_i = S_{i,1} \cup \dots \cup S_{i,k}$ (line 22). To obtain $S_{i,j}$, the algorithm first computes the set St of states in $G_{type(n_p)}$ that the nodes in P_j belong to (line 18) (by the definition of $T(a, r)$, for a node n and its parent n_p label $l(n)$ must be a state in $G_{type(n_p)}$). Then it finds the set St' of states reachable from a state in St in $G_{type(n_p)}$ via location step $\rightarrow^+ :: lb[i]$ (line 19). Finally, the set $S_{i,j}$ of nodes belonging to a state in St' is obtained (line 20).

Input: A query $q = /ax[1] :: lb[1]/\dots/ax[m] :: lb[m]$ in $XP^{\{\downarrow, \downarrow^*, \uparrow, \uparrow^*, \rightarrow^+, \leftarrow^+\}}$ and a fixed nonrecursive DTD $D = (d, s)$.

Output: “yes” or “no”.

begin

1. Compute $T(s)$.
2. **for** each $t' \in T(s)$ **do**
3. $t \leftarrow n_{root}(t')$ with $l(n_{root}) = root$; // n_{root} is the “dummy” root of t
4. $S_0 \leftarrow \{n_{root}\}$;
5. **for** $i = 1$ **to** m **do**
6. **if** $ax[i] = \downarrow$ **then**
7. $S_i \leftarrow \{n \mid n \text{ is a child of a node in } S_{i-1}, l(n) \stackrel{\#}{=} lb[i]\}$;
8. **else if** $ax[i] = \downarrow^*$ **then**
9. $S_i \leftarrow \{n \mid n \text{ is a descendant of a node in } S_{i-1}, l(n) \stackrel{\#}{=} lb[i]\}$;
10. **else if** $ax[i] = \uparrow$ **then**
11. $S_i \leftarrow \{n \mid n \text{ is the parent of a node in } S_{i-1}, l(n) \stackrel{\#}{=} lb[i]\}$;
12. **else if** $ax[i] = \uparrow^*$ **then**
13. $S_i \leftarrow \{n \mid n \text{ is an ancestor of a node in } S_{i-1}, l(n) \stackrel{\#}{=} lb[i]\}$;
14. **else if** $ax[i] = \rightarrow^+$ **then**
15. Find the s-partition of S_{i-1} . Let P_1, \dots, P_k be the result.
16. **for** $j = 1$ **to** k **do**
17. Let n_p be the parent of P_j ;
18. $St \leftarrow \{l(n) \mid n \in P_j\}$;
19. $St' \leftarrow \{q^h \mid q^h \text{ is reachable from a state in } St \text{ in } G_{type(n_p)}, (q^h) \stackrel{\#}{=} lb[i]\}$;
20. $S_{i,j} \leftarrow \{n \mid n \text{ is a child of } n_p \text{ in } t, l(n) \in St'\}$;
21. **end**
22. $S_i \leftarrow S_{i,1} \cup \dots \cup S_{i,k}$;
23. **else if** $ax[i] = \leftarrow^+$ **then**
24. Find the s-partition of S_{i-1} . Let P_1, \dots, P_k be the result.
25. **for** $j = 1$ **to** k **do**
26. Let n_p be the parent of P_j ;
27. $St \leftarrow \{l(n) \mid n \in P_j\}$;
28. $St' \leftarrow \{q^h \mid St \text{ contains a state reachable from } q^h \text{ in } G_{type(n_p)}, (q^h) \stackrel{\#}{=} lb[i]\}$;
29. $S_{i,j} \leftarrow \{n \mid n \text{ is a child of } n_p \text{ in } t, l(n) \in St'\}$;

```

30.     end
31.      $S_i \leftarrow S_{i,1} \cup \dots \cup S_{i,k};$ 
32.     end
33. end
34. if  $S_m \neq \emptyset$  then return “yes”;
35. end
36. return “no”;
     end

```

Example 3. Consider executing the algorithm for $q = / \downarrow :: s / \downarrow :: b / \rightarrow^+ :: c / \downarrow :: e$ and the DTD in Example 2. For t_1 and t_2 in Fig. 4(a), we have $S_0 = \{n_{root}\}$, $S_1 = \{n_1\}$, and $S_3 = \emptyset$. For t_3 , we have $S_0 = \{n_{root}\}$, $S_1 = \{n_1\}$, and $S_2 = \{n_2\}$. Consider S_3 . In lines 14 to 22, since the s-partition of S_2 is $P_1 = \{n_2\}$, we have $St = \{l(n_2)\} = \{b^4\}$ and $St' = \{c^6\}$ according to G_{bc^*} (Fig. 4(b)). Thus $S_3 = S_{3,1} = \{n_3, n_5\}$, but $S_4 = \emptyset$ by line 7 since $l(n_3) \neq e$ and $l(n_5) \neq e$. Hence q is unsatisfiable under D .

The correctness of the algorithm can be shown by induction on the nesting level of ‘*’ operators in D (the detail are omitted because of space limitation). As for the time complexity of the algorithm, since D is a nonrecursive fixed DTD, it is clear that the size of each tree $t \in T(s)$ in line 2 is fixed and $|T(s)|$ is a fixed number. This implies that lines 5 to 33 can be done in $O(m)$ time. Thus we have the following.

Theorem 3. $\text{SAT}(\text{XP}^{\{\downarrow, \downarrow^*, \uparrow, \uparrow^*, \rightarrow^+, \leftarrow^+\}})$ is solvable in linear time under fixed nonrecursive DTDs.

Finally, let us consider relaxing the “nonrecursive” restriction in Theorem 3. We focus on the fact that for a recursive DTD $D = (d, s)$ and a query $q \in \text{XP}^{\{\downarrow, \downarrow^*, \uparrow, \uparrow^*, \rightarrow^+, \leftarrow^+\}}$, q does not always visits recursive elements of D . Consider the graph representation of a DTD (Fig. 5). For labels a, b , b is a descendant of a in D if $\text{dist}(s, a) < \text{dist}(s, b)$, where $\text{dist}(s, a)$ is the (shortest) distance between s and a in the graph of D . A pair (a, b) of labels is recursive if b is a descendant of a and a occurs in $d(b)$. For a recursive pair (a, b) , let

$$R(a, b) = \{a\} \cup \{c \mid c \text{ is a descendant of } a \text{ but not of } b\}.$$

We define that

$$R(D) = \bigcup_{(a, b) \text{ is a recursive pair of } D} R(a, b).$$

For example, in Fig. 5(b) (b, e) is the only recursive pair of D and $R(D) = R(b, e) = \{b, e, f\}$. By definition the subgraph consisting of the labels in $\Sigma \setminus R(D)$ is acyclic.

Let $D = (d, s)$ be a DTD and $q = /ax[1] : lb[1] / \dots / ax[m] :: lb[m]$ be a query such that every upward axis of q visits only labels in $\Sigma \setminus R(D)$. Then the satisfiability of q under D can be checked as follows.

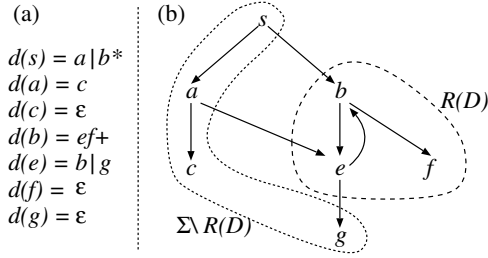


Fig. 5. (a) DTD $D = (d, s)$ and (b) A graph representation of D

1. If $lb[1] \in \Sigma \setminus R(D)$, then goto step 2. Otherwise, goto step 3.
2. Run the algorithm in this section for input (q, D) . While running the algorithm, if the following two conditions hold, then the context node moves from $\Sigma \setminus R(D)$ to $R(D)$, thus set $q = /ax[i + 1] : lb[i + 1]/ \cdots /ax[m] :: lb[m]$ and $D = (d, lb[i + 1])$ and goto step 3.
 - The current label $lb[i] \in \Sigma \setminus R(D)$ but the next label $lb[i + 1] \in R(D)$.
 - If there is an index $k \geq 2$ such that $ax[i + 2], \dots, ax[i + k]$ are all sibling axes but that $ax[i + k + 1]$ is not a sibling axis, then $lb[i + k] \in R(D)$.
3. Run the algorithm in the previous section for input (q, D) . While running the algorithm, if the current label $lb[i] \in R(D)$ but the next label $lb[i + 1] \in \Sigma \setminus R(D)$, then set $q = /ax[i + 1] : lb[i + 1]/ \cdots /ax[m] :: lb[m]$ and $D = (d, lb[i + 1])$ and goto step 2.

It is clear that the above method runs in polynomial time. We have the following.

Theorem 4. *Let D be a fixed DTD and $q = /ax[1] :: lb[1]/ \cdots /ax[m] :: lb[m] \in \text{XP}\{\downarrow, \downarrow^*, \uparrow, \uparrow^*, \rightarrow^+, \leftarrow^+\}$ be a query. If one of the following conditions hold, then the satisfiability of q under D can be determined in polynomial time.*

- D is nonrecursive.
- For every location step $ax[i] :: lb[i]$ of q , if $ax[i] \in \{\uparrow, \uparrow^*\}$, then neither $lb[i - 1]$ nor $lb[i]$ is in $R(D)$.

6 Conclusion

In this paper, we have considered the XPath satisfiability problem under fixed DTDs. We first have shown that $\text{SAT}(\text{XP}\{\downarrow, \uparrow\})$ is NP-complete under fixed DTDs. We have next shown that $\text{SAT}(\text{XP}\{\downarrow, \downarrow^*, \rightarrow^+, \leftarrow^+\})$ is in PTIME under fixed DTDs. Finally, we have presented sufficient conditions under which $\text{SAT}(\text{XP}\{\downarrow, \downarrow^*, \uparrow, \uparrow^*, \rightarrow^+, \leftarrow^+\})$ is in PTIME under fixed DTDs.

There are many things to do as future works. First, the XPath fragments considered in this paper are restricted in the sense that only a label is allowed

as a node test and that neither union nor qualifier is supported. Therefore, we have to explore more general XPath fragments for which satisfiability is tractable under fixed DTDs. Second, we would like to consider more powerful schema languages such as XML Schema and RELAX NG. In particular, the algorithms in Sections 4 and 5 requires several modifications so that they handle element types supported by such schema languages. On the other hand, the NP-completeness shown in Sect. 3 still holds under such schema languages. Finally, the algorithms shown in this paper has not been implemented yet. We need to implement them and make experiments on the efficiency and the availability of the algorithms.

Acknowledgement. The author would like to thank Prof. George H. L. Fletcher for discussions on this topic.

References

1. Benedikt, M., Fan, W., Geerts, F.: Xpath satisfiability in the presence of dtds. *Journal of the ACM* 55(2) (2008)
2. Brüggemann-Klein, A., Wood, D.: One-unambiguous regular languages. *Information and Computation* 142(2), 182–206 (1998)
3. Figueira, D.: Satisfiability of downward xpath with data equality tests. In: *Proc. PODS*, pp. 197–206 (2009)
4. Geerts, F., Fan, W.: Satisfiability of xpath queries with sibling axes. In: Bierman, G., Koch, C. (eds.) *DBPL 2005. LNCS*, vol. 3774, pp. 122–137. Springer, Heidelberg (2005)
5. Hidders, J.: Satisfiability of xpath expressions. In: Lausen, G., Suciu, D. (eds.) *DBPL 2003. LNCS*, vol. 2921, pp. 21–36. Springer, Heidelberg (2004)
6. Ishihara, Y., Shimizu, S., Fujiwara, T.: Extending the tractability results on xpath satisfiability with sibling axes. In: Lee, M.L., Yu, J.X., Bellahsene, Z., Unland, R. (eds.) *XSym 2010. LNCS*, vol. 6309, pp. 33–47. Springer, Heidelberg (2010)
7. Lakshmanan, L.V.S., Ramesh, G., Wang, H., Zhao, Z.J.: On testing satisfiability of tree pattern queries. In: *Proc. VLDB*, pp. 120–131 (2004)
8. Montazerian, M., Wood, P.T., Mousavi, S.R.: Xpath query satisfiability is in ptime for real-world dtds. In: Barbosa, D., Bonifati, A., Bellahsene, Z., Hunt, E., Unland, R. (eds.) *XSym 2007. LNCS*, vol. 4704, pp. 17–30. Springer, Heidelberg (2007)
9. Suzuki, N.: An algorithm for inferring k optimum transformations of xml document from update script to dtd. *IEICE Trans. Inf. & Syst.* E93-D(8), 2198–2212 (2010)
10. Suzuki, N., Fukushima, Y.: Satisfiability of simple xpath fragments in the presence of dtds. In: *Proc. WIDM*, pp. 15–22 (2009)

Computing Compressed XML Data from Relational Databases

Stefan Böttcher, Dennis Bokermann, and Rita Hartel

University of Paderborn, Computer Science, Fürstenallee 11, 33102 Paderborn, Germany
{stb@,dbn@mail.,rst@}uni-paderborn.de

Abstract. SQL/XML allows generating an XML document as the result of a query that is evaluated on relational data. This facilitates companies sharing their relational data in form of XML documents with other companies or with a marketplace, but significantly increases the exchanged data volume. To reduce both the volume of the exchanged data by exchanging compressed XML and the time needed for compression, we propose an approach that allows preparation of a compressed XML document as the answer to an SQL/XML query directly, i.e., without the need to create the XML document first and compress it afterwards. Our evaluation has shown that generating the compressed document directly is in most cases faster than generating the uncompressed XML document and compressing it, and in some cases it is even faster than the generation of the uncompressed XML document alone. As our approach of generating compressed XML requires only SQL support from the underlying database system, a second advantage is that it can be used for the generation of compressed XML even for database systems that do not (yet) support SQL/XML (like MySQL).

Keywords: XML compression, relational Databases, SQL/XML.

1 Introduction

1.1 Motivation

Since SQL:2003, SQL is extended by SQL/XML to allow use of XML in conjunction with SQL. The SQL/XML specification includes functions to construct XML data. These functions allow the user to construct new XML elements or attributes with text values or attribute values taken for example from relational tables. Other functions such as XMLCONCAT or XMLAGG can be used for combining small XML fragments into larger ones. Therefore, SQL/XML makes it possible to create XML documents as the result of a query which is evaluated on a relational database.

Typical applications of SQL/XML are cross-company applications where companies transform parts of their relational data into XML and send the transformed XML data to a second company, to a customer or to a marketplace. While the XML format due to its flexibility is a popular data format, the overhead caused by this flexible structure is the biggest disadvantage of using XML as data exchange format. Exchanging a compressed XML representation instead of uncompressed XML might

help to solve this problem. But while sending compressed XML will lead to less data transfer and to lower data transfer time, taking the indirection from relational data to XML and then to compressed XML might lead to additional computational effort on the sender's side. This additional effort can be avoided, when compressed XML is directly generated from an SQL/XML statement.

1.2 Contributions

In this paper, we propose an approach to generate compressed XML data directly from relational data as the result of an SQL/XML query. We describe a performance evaluation that shows that this leads to faster data transfer (i.e., a speed-up by a factor of 4 to 8 with our test data) and to less data volume without the additional computational effort caused by the indirection.

As we transform the SQL/XML query into an SQL query to retrieve the data from the relational database which is used to compute the compressed XML result, a second advantage of our approach is that we do not require the database system to support SQL/XML queries. SQL support is sufficient for our approach, such that it can be used with database systems that do not (yet) support SQL/XML (like MySQL).

1.3 Paper Organization

The paper is organized as follows: Section 2 describes the main idea of our approach, introduces SQL/XML and the example being used in the remainder of this paper, and explains the main idea of our approach. Section 3 gives an overview of how the XML schema of the SQL/XML query result and the compressed data can be derived from the given SQL/XML query and the relational database. Section 4 describes a performance evaluation of the compression ratio, the computation time, and the transfer time of our approach. Section 5 compares our approach to related work. Finally, Section 6 summarizes our contributions.

2 The Concept

2.1 Considered Subset of SQL/XML

In this paper, we only consider the “raw” XML data and ignore additional XML constructs like comments or processing instructions. Therefore, we can restrict the XML publishing functions of SQL/XML to the functions described in Table 1 in combination with the SQL commands SELECT, FROM, WHERE and ORDER BY. We allow the Queries to be nested at any extend.

The example being used in this paper for explaining our approach involves a company database containing items, parts, orders, suppliers, and customers. On the left hand side of Table 2, we can see an SQL/XML query that, applied to our database, returns the XML document outlined on the right-hand side of Table 2.

Table 1. XML publishing functions of SQL/XML

XMLELEMENT	Creates an XML element. The XMLELEMENT function has the element name as first parameter, the attribute list, i.e. an XMLATTRIBUTES function call, as an optional second parameter and the content as a list of optional further parameters.
XMLATTRIBUTES	Creates XML attributes. The XMLATTRIBUTES function has a list of attributes as parameters, where the first part of each attribute is the column name from where to read the attribute value and the second part is the attribute name.
XMLFOREST	Creates a forest of XML. The XMLFOREST function has a list of its contents as parameters, where the first part of each content is the content definition and the second part is the label of the tag by which the content is surrounded.
XMLCONCAT	Combines a list of individual XML values to create a single value containing an XML forest. The XMLCONCAT function has a list of its contents as parameters and yields one result row per tuple to which it is applied.
XMLAGG	Combines a collection of rows, each containing a single XML value, to create a single value containing an XML forest. The XMLAGG function has a call to an XML publishing function as first parameter and an 'order by' clause that defines the order of its content as optional second parameter. The XMLAGG function call results in a single result row containing the whole forest.

Table 2. SQL/XML query and query result of our example

<pre> (1) SELECT (2) XMLFOREST((3) XMLAGG ((4) XMLELEMENT(NAME nation, (5) XMLATTRIBUTES ((6) n.N_NAME AS name), (7) (SELECT XMLAGG ((8) XMLELEMENT ((9) NAME customer, (10) c.C_NAME) (11) ORDER BY c.C_NAME (12)) FROM CUSTOMER c (13) WHERE c.C_NATIONKEY = (14) n.N_NATIONKEY) (15)) ORDER BY n.N_NATIONKEY (16)) AS costumers (17)) (18) FROM NATION n (19) WHERE n.N_NATIONKEY < 5 </pre>	<pre> <costumers> <nation name="Germany"> <customer>C#007</customer> <customer>C#013</customer> </nation> <nation name="France"> <customer>C#042</customer> </nation> <nation name="USA"></nation> <nation name="Spain"> <customer>C#023</customer> <customer>C#101</customer> </nation> </costumers> </pre>
---	--

2.2 This Paper’s Example

The XMLFOREST function call creates the customers element and the XMLAGG function call nested inside aggregates the result of the query that is the parameter of XMLAGG into an XML forest in the order given by the order by clause. The XMLELEMENT and XMLATTRIBUTE function calls generate an XML element or an XML attribute respectively with the given name and the specified content.

2.3 The Basic Idea

In order to generate compressed XML directly from an SQL/XML query and the contents of a relational database, we extend the idea behind the compression technique of XSDS [1] as follows. While XSDS removes those XML nodes, the

existence of which can be deduced from an XML schema, we do not even generate these XML nodes of an SQL/XML query that would be removed later by XML compression. In XSDS [1], these non-generated nodes are the XML nodes predefined by an XML schema definition, whereas here, these non-generated nodes are the XML nodes generated by the SQL/XML query. Omitting these nodes will significantly reduce the overhead of the XML data generated by the SQL/XML query from the relational data. Note that the compressed XML document can be decompressed to XML on demand, and XPath queries can be answered directly on the compressed document by the XSDS decompressor described in [1].

The result of an SQL/XML query is a single XML document consisting of the XML structure (i.e. the element tags and attributes) on the one hand and the text data (i.e. the text and attribute values) on the other hand. We store all text values in document order in a single text container that is compressed via gzip.

Concerning the XML document structure, we store the ‘fixed’ part of the structure, i.e., that part that can be derived from the query without any knowledge on the data, within an XML schema. For the example given in Table 2, we know e.g. that each element with label ‘nation’ has an attribute with attribute name ‘name’ and it contains any number of elements with label ‘customer’. If the SQL/XML query is known at the receiver’s side, the schema could even be generated there, i.e., there is no need to transfer the schema. In addition to the fixed part of the document that is stored in the XML schema, we need to store the variant parts of the XML document. Whenever there is a nested query or whenever there is a call to the function XMLAGG, the number of elements to be created depends on the data stored in the relational database, i.e., the number varies. Therefore, we have to store the number of occurrences in our compressed data format for the elements generated by a nested query or by the function XMLAGG.

We use the following two steps to generate the compressed XML data as a result for an SQL/XML query:

In the first step, we analyze the SQL/XML query to compute a set of templates that are repeated substructures within the compressed document’s structure. Therefore, we compute the set of templates in the form of an XML schema of the result document based on the SQL/XML query alone.

In the second step, we query the relational data to examine, how the templates that were generated in the first step have to be combined to form the complete structure of the document. We do this by constructing an SQL query that retrieves the text values, and from which the compressed document structure can be computed. At the same time, we use the results of our query to compute the text values of the result document.

The output of these two steps is an XML schema on the one hand, and the compressed XML document containing the document structure in compressed format and the compressed text values on the other hand.

3 Retrieving Compressed XML Data from a Relational Database

3.1 Generating the XML Schema for the SQL/XML Query Result

In this first step, we analyze the SQL/XML query and compute the XML schema according to which the resulting document will be valid. For the schema generation,

we can use a construction kit that provides a schema template for each SQL/XML function. The nested structure of the functions within the SQL/XML query determines the nested structure of the schema elements.

Fig. 1 shows the XML schema generated for the SQL/XML query of Table 2.

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="costumers">
    <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="nation">
          <xsd:complexType>
            <xsd:sequence minOccurs="0" maxOccurs="unbounded">
              <xsd:element name="customer" type="xsd:string"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

Fig. 1. Schema derived from the query of Table 2

For each call of the XMLELEMENT function, we generate an `<xsd:element>`-element with the given name within the schema. If the XMLELEMENT function does not define any element content, i.e., it defines an empty element, the `<xsd:element>`-element has an empty `<xsd:complexType>`-element as single child node. If the content of the XMLELEMENT is the value of a database column, i.e., the content is a text node, the `<xsd:element>`-element is empty and contains an attribute `type="TYPE"`, where TYPE is e.g. `xsd:string` or `xsd:int`, depending on the data type of the given database column. Finally, if the XMLELEMENT function has a more complex content CC as third parameter (e.g. other XML elements or attributes), the `<xsd:element>`-element contains an `<xsd:complexType>`-element as child node that itself contains data that follow an XML schema fragment that corresponds to CC.

For each parameter of an XMLATTRIBUTES function that itself is contained in the second parameter of an XMLELEMENT function EL, we generate an `<xsd:attribute>`-element with the given name and the attribute `use="required"` and embed it into the `complexType` that is the child node of the `<xsd:element>`-element within the schema that corresponds to EL.

Similarly, for each call of the XMLFOREST function with parameters p_1, \dots, p_n , we generate an `<xsd:sequence>`-element that contains an `<xsd:element>`-element with the name `label(pi)` for each of the parameters p_i , where `label(p)` represents the label part of the parameter p of the XMLFOREST function.

For each call of the XMLCONCAT function, we generate an `<xsd:sequence>`-element, the content of which corresponds to the content of the XMLCONCAT function.

For each call of the XMLAGG function, we generate an `<xsd:sequence>`-element that has an attribute `minOccurs="0"` and an attribute `maxOccurs="unbounded"` and that contains the content that corresponds to the content of the XMLAGG function.

Finally, for each embedded SQL/XML query that does not contain a call to an XMLAGG function, we create an `<xsd:sequence>`-element that has attributes `minOccurs="0"` and `maxOccurs="1"`, as each SQL/XML query either returns an empty

result or a single XML document. The created element contains the content that corresponds to the content that is created as a result of the embedded SQL/XML query.

3.2 Computing the SQL Query for Generating the Compressed XML Document

After creating the XML schema based on the SQL/XML query, we have to look up the data to create the compressed structure and the compressed text data. Therefore, we transform the given SQL/XML query SX into an SQL query SQ that returns all text values of the resulting XML document in document order. At the same time, we enrich SQ such that it returns additional data that helps to assign the tuples TSQ of SQ's result to that sub-query of SQ which TSQ corresponds to.

For example, Fig. 2 shows the SQL query derived from the SQL/XML query given in Table 2.

(1) SELECT	(9) SELECT
(2) 1 AS QueryID,	(10) 2 AS QueryID,
(3) "n"."N_NAME" AS COL_1_1,	(11) "n"."N_NAME" AS COL_1_1,
(4) "n"."N_NATIONKEY" AS COL_1_2,	(12) "n"."N_NATIONKEY" AS COL_1_2,
(5) null AS COL_2_1	(13) "c"."C_NAME" AS COL_2_1
(6) FROM "NATION" "n"	(14) FROM "NATION" "n", "CUSTOMER" "c"
(7) WHERE "n"."N_NATIONKEY" < 5	(15) WHERE "n"."N_NATIONKEY" < 5 AND
(8) UNION ALL	(16) "c"."C_NATIONKEY"="n"."N_NATIONKEY"
	(17) ORDER BY COL_1_2, COL_1_1, COL_2_1

Fig. 2. SQL query derived from the SQL/XML query of Table 2

In the best case, the result of the computed SQL query SQ not only returns the text values of the resulting XML document in document order, but it also reflects the nesting of the XML elements, the text values refer to.

In order to derive this information, we compute the query given in Fig. 2 which has the result shown in Table 3.

The query result in Table 3 is being used as follows. The first column is the query ID to which the tuple belongs to. Query_ID 1 refers to the outer query (lines 3-16) of the SQL/XML query shown in Table 2 and Query ID2 refers to the inner query (lines 7-14) of the SQL/XML query shown in Table 2. By splitting the results of Table 3 at each tuple with query ID "1" and building m=4 groups of n tuples with query ID 2 where n ≥ 0, we can derive the required information on the nesting of the XML elements. The value in column COL_1_1 builds the N_NAME of each group, the value in column COL_1_2 the N_NATIONKEY, the size n of each group is the value of COUNT and the values in COL_2_1 build the values of the column C_NAME.

Given an SQL/XML query SX, we compute the corresponding SQL query Q as follows:

Whenever SX does not contain embedded queries and only contains a call to the XMLAGG function, we can simply concatenate columns that are referred to within SX (as output columns or as columns within the ORDER BY-clause) to the SELECT-clause of SQ. Furthermore, we can adopt the FROM- and the WHERE-clauses of SQ from the FROM- and the WHERE-clauses of SX, and we can adopt the ORDER BY-clause of SQ from the XMLAGG function.

Table 3. Result of the SQL query given in Fig. 2

Query_ID	COL_1_1	COL_1_2	COL_2_1
2	Germany	1	C#007
2	Germany	1	C#013
1	Germany	1	(null)
2	France	2	C#042
1	France	2	(null)
1	USA	3	(null)
2	Spain	4	C#023
2	Spain	4	C#101
1	Spain	4	(null)

Whenever *SX* does not contain embedded queries, but contains $n > 1$ calls to the *XMLAGG* function, we generate n queries as described above, where each query considers only a single *XMLAGG* function and ignores all the others. These queries are later combined to a single query by computing the union of all these queries.

For each embedded query *QE* that is embedded into an outer query *QO* of *SX*, we generate a separate SQL query *EQ*. The *SELECT*-clause of *EQ* contains the concatenation of all columns referred to in *QO* followed by all columns referred to in *QE*. Furthermore, we concatenate the *FROM*-clauses of *QO* and *QE* to the *FROM*-clause of *EQ*, and we concatenate the *WHERE*-clauses *WQO* of *QO* and *WQE* of *QE* to the *WHERE*-clause ‘*WQO AND WQE*’ of *EQ*. Finally, the *ORDER BY*-clause of *EQ* contains the *ORDER BY*-clause of *QO* followed by the *ORDER BY*-clause of *QE*. For example, let *QE* be the inner query given in lines (7)-(14) of Table 2, and let *QO* be the complete query given in Table 2. Then, the query *EQ* that corresponds to *QE* is given in lines (9)-(16) of Fig. 2, where *COL_1_1* and *COL_1_2* are the columns referred to in *QO* and *COL_2_1* is the column referred to in *QE*, the *FROM*-clause contains the tables *NATION* of *QO* and *CUSTOMER* of *QE*, and the *WHERE*-clause is a conjunction of the *WHERE*-clauses of *QO* (line (19) of Table 2) and of *QE* (line (13) of Table 2).

In order to allow the query optimizer to avoid querying the same sub-queries several times, we combine all created queries into a single query by building the union of all the queries. In order to avoid ambiguities within the combined query, we rename the columns of the queries in such a way, that they all carry disjoint column names. Only if a query *QE* was embedded into a query *QO* and therefore ‘inherits’ the join-columns from *QO*, the names of the columns of *QO* occur in *QO* as well as in *QE*. Furthermore, we insert empty columns to each query such that all query results contain the same set of columns and we insert a constant column *QueryID* to each query that helps to assign the result tuples to the original query which they stem from.

Finally, the *ORDER BY*-clauses of all queries are concatenated to the global *ORDER BY*-clause of the *UNION*-query, such that the *ORDER BY*-clause of the outer query occurs before the *ORDER BY*-clause of the inner query and the *ORDER BY*-clause of a query *Q1* that occurs before a query *Q2* occurs before the *ORDER BY*-clause of *Q2*. Finally, the *ORDER BY*-clause that has been adopted from an *XMLAGG* function call within a query occurs before all other columns of this query within the global *ORDER BY*-clause. For example, the *ORDER BY*-clause of the SQL query shown in Fig. 2 first contains the column *COL_1_2*, which corresponds to the column *N_NATIONKEY* of the *ORDER BY* of line (15) of Table 2, followed by

COL_1_1 (corresponding to N_NAME of the outer query) and finally followed by COL_2_1 (corresponding to the column C_NAME of the inner query).

3.3 Computing the Compressed Structure and the Text Data of the Compressed XML Document

Example. Fig. 3 shows the resulting compressed document for the query of our example. The structure starts with a ‘4’ (there are 4 nation tags) followed by a ‘2’ (there are 2 customers in Germany), a ‘1’ (there is 1 customer in France), a ‘0’ (there are 0 customers in USA) and finally by a ‘2’ (there are 2 customers in Spain). The text data contains the text nodes (and attribute values) of the result document shown in Table 2 in document order.

```
Structure: 4 2 1 0 2
Text data: Germany C#007 C#013 France C#042 USA Spain C#023 C#101
```

Fig. 3. Compressed document of our example

Overview. In order to compute the compressed document, first, the SQL query given in Fig. 2 is evaluated and returns the results shown in Table 3.

The ORDER BY-clause of SQ is constructed in such a way that the text values of the result to the SQL/XML query SX occur in the result to SQ in document order (c.f. tables 3 and 4). Whenever the results of a sub-query SX1 of SX occur before the results of a sub-query SX2 of SX in the result document, the columns of the sub-query Q1 of SQ that corresponds to SX1 occur before the columns of the sub-query Q2 of SQ that corresponds to SX2 in the ORDER BY-clause of SQ, and both sub-queries have disjoint sets of columns. Therefore, in the result set of SQ, all tuples of Q1 occur before the tuples of Q2 (as the null value is considered as being the greatest value). That means, that we can consider the tuples of Q1 independently of the tuples of Q2 and compute the text values and the compressed structure independently of each other and concatenate them to the compressed result afterwards.

Whenever the result of a sub-query SX2 of SX is contained within the result of a sub-query SX1 of SX in the result document, the tuples of the sub-query Q2 of SQ that corresponds to SX2 as well as the tuples of the sub-query Q1 of SQ that corresponds to SX1 contain the columns and values of Q1, but only the tuples of Q2 contain the columns and values of Q2 and the tuples of Q1 contain null values in these columns. The ORDER BY-clause of the query contains the columns of Q1 before the columns of Q2. Therefore, for each value returned by sub-query Q1 for an XML element, the result set contains each tuple returned by sub-query Q2 for a nested XML element in the XML file, followed by the tuple of sub-query Q1 (c.f. Table 3).

Retrieving the Text Data. We can split each sub-sequence of tuples within the result set belonging to two sub-queries Q1 and Q2, where Q1 contains Q2, into several sub-sequences SSQ, whereas each sub-sequence SSQ ends with a line that has the Query_ID of Q1 and contains the same values for the columns belonging to Q1.

For each such sub-sequence SSQ containing n tuples, we add the following to the list of text values: First, the values of the columns of the first tuple belonging to output columns of Q1 (i.e., the values of the outer query), followed by the values of the

columns of tuples 1 to n-1 belonging to the output columns of Q2 (i.e., the values of all tuples of the inner query). If we consider the sub-sequence SSQ consisting of the tuples 7-9 of Table 4, we first write the value 'Spain' (i.e., the value of tuple 7 – the first tuple - belonging to the output columns of Q1) followed by the values C#023 and C#101 (i.e., the value of tuples 7 and 8, which are the 1..n-1 values for size n=3 of SSQ consisting of the tuples 7-9).

Retrieving the Compressed Document Structure. Let Q1 and Q2 be two nested sub-queries, where Q1 contains Q2. At the beginning of each sub-sequence of tuples within the result set belonging to Q1, we store a counter in the structure stream that is initialized with 0 and that is incremented, whenever a tuple with the Query_ID of Q1 is read. Similarly, at the beginning of each nested sub-sequence of tuples within the result set belonging to Q2, we add a new counter cQ2 in the structure stream that is initialized with 0 and that is incremented, whenever a tuple with Query_ID of Q2 is read. The counter cQ2 is closed (i.e., no more incrementation is possible), whenever a tuple with Query_ID of Q1 is read.

If we apply this process to the query result shown in Table 3 of the query given in Fig. 2, we get exactly the compressed XML document as given in Fig. 3.

Remember that decompression back to XML and querying the compressed document can be done by the XSDS decompressor described in [1].

4 Evaluation

We have evaluated our approach using the database systems Oracle 10g Express and IBM DB2 Express. As both have shown similar results, we concentrate on the DB2 results within this evaluation section.

We have used the TCP-H benchmark (<http://www.tpc.org/tpch/>) to create a relational database. We have tested 5 different kinds of queries, that select customers sorted by nation (CN4 and CN16), customer data (C400 and C3200), article data (A4 and A16), supplier data (S4 and S16) and order data including customer and supplier information (O2 and O4). Each of these queries contain a range clause within the where clause, such that the result size can be scaled.

For the evaluation of the compression ratio reached by XSDS, please refer to [1].

Fig. 4 shows the query evaluation times for our set of queries for the indirect approach (i.e., evaluating SQL/XML query and then compressing the result) on the one hand and for the direct approach (generating compressed XML directly from the SQL/XML query and the relational data) on the other hand in relation to the SQL/XML query evaluation time (100%). We can see that our approach not only takes less time to compute the compressed data directly than the total time of the indirect approach, but that for all queries tested, it can even directly compute the compressed data in less time than the SQL/XML query evaluation alone takes. Furthermore, we can see that our approach scales better for larger result sets, as for each pair of queries that carry the same initial letters, where the result size was scaled up (e.g. S4 and the 4 times larger S16), we can see that the performance gain compared to the query evaluation time is better, when the result gets larger.

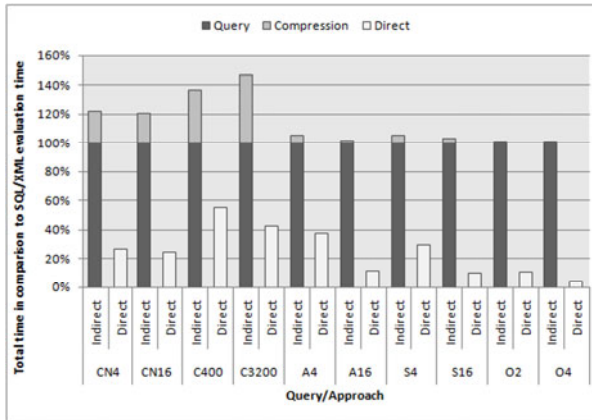


Fig. 4. Evaluation times for the indirect and the direct approach

Fig. 5 shows a simulation of a scenario, where the query result is computed on the sender’s side and then transferred to a receiver. Fig. 5 shows the total time for the indirect and the direct approach. Here, the total time t_i , consumed for the indirect approach consists of SQL/XML query evaluation + compression + transfer time of schema and the compressed document. Furthermore, the total time t_d consumed for the direct approach consists of schema generation + SQL query generation + evaluation + transfer time of schema and the compressed document. Finally, the total time t_u consumed for the uncompressed approach consists of query evaluation + transfer time of the uncompressed document. The times retrieved for all queries are summed up for all documents. Fig. 5 shows the values t_i/t_u and t_d/t_u .

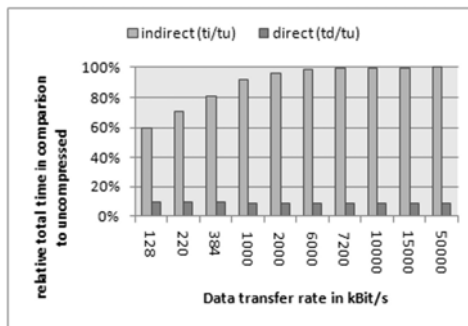


Fig. 5. Total time (computation plus transfer) consumed by the direct approach and the indirect approach in comparison to uncompressed transfer of the results for different data rates

While the indirect approach only outperforms the uncompressed approach for low data transfer rates, the direct approach constantly needs only about 10% of the time that the uncompressed approach needs.

In client-server scenarios where the query is generated on the client side, it is not even necessary to send the schema back to the client, as it can be derived from the client's query, i.e., we expect our approach to get an even higher data transfer rate.

5 Related Work

For a given SQL/XML query, our approach calculates the schema of the result document as well as an SQL query that retrieves all the data that is needed to compute a compressed representation of the result document. Then, both are used for computing the compressed result document.

Approaches that also try to build a bridge from the relational world to the XML world are the approaches [2], [3] that translate SQL queries into XQuery expressions that query the same data or that provide SQL-based access to relational data via relational views over XML data [4]. The other direction of the bridge, i.e., XML views over relational data is discussed in [5], [6], or storing XML data in form of relational tables is discussed in [7], [8], [9].

The problem of deriving schema information not for a given query but for a collection of documents is discussed for DTDs in [10].

None of these approaches are being used for directly transforming SQL/XML queries in such a way to SQL queries that the SQL query result can be used for generating compressed XML instead of generating XML. To the best of our knowledge, our approach is the first that provides this property.

6 Conclusions

In this paper, we have presented an approach that allows generating a compressed XML document directly as the result of an SQL/XML query that is applied to a relational database. In contrast to computing the uncompressed XML query result first and compressing it as a second step, our approach yields the compressed data in less time. In contrast to only creating the uncompressed XML document and sending it in uncompressed format, creating the compressed XML data directly and sending it in compressed format takes total times of 23% down to 12% of the time needed to create and transfer the uncompressed XML depending on the available data rate.

As today only a few DBMSs (e.g. Oracle 11g and IBM DB2) support the evaluation of SQL/XML queries, while the majority of DBMSs do not yet support SQL/XML, a further advantage of our approach is that it requires only SQL support. This means that in addition to DBMSs that support SQL/XML all other DBMSs that only support SQL can be used to produce the compressed XML document as a result to an SQL/XML query. Together with our XSDS decompressor, this technique can even be used as a fairly fast substitute for an SQL/XML implementation that supports a subset of SQL/XML, as our generation of compressed XML together with an additional decompression step using our XSDS decompressor at least for some SQL/XML queries outperforms the DB2 and the Oracle SQL/XML implementation.

Our approach contains parts that are generic, while other parts are specific for the XML compression approach being used. By changing the parts that are specific for

the XML compression approach being used, the approach could be applied to other compression approaches as well. Therefore, it might be an interesting task to integrate different XML compression techniques and to compare and to evaluate the times to create the compressed data and to transfer the compressed data to the receiver.

References

1. Böttcher, S., Hartel, R., Messinger, C.: Searchable Compression of Office Documents by XML Schema Subtraction. In: Lee, M.L., Yu, J.X., Bellahsene, Z., Unland, R. (eds.) XSym 2010. LNCS, vol. 6309, pp. 103–112. Springer, Heidelberg (2010)
2. Jahnkuhn, H., Bruder, I., Balouch, A., Nelius, M., Heuer, A.: Query Transformation of SQL into XQuery Within Federated Environments. In: Grust, T., Höpfner, H., Illarramendi, A., Jablonski, S., Fischer, F., Müller, S., Patranjan, P.-L., Sattler, K.-U., Spiliopoulou, M., Wijssen, J. (eds.) EDBT 2006. LNCS, vol. 4254, pp. 577–588. Springer, Heidelberg (2006)
3. Jigyasu, S., Banerjee, S., Borkar, V., Carey, M., Dixit, K., Malkani, A., Thatte, S.: SQL to XQuery Translation in the AquaLogic Data Services Platform. In: Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, Atlanta, GA, USA, p. 97 (2006)
4. Halverson, A., Josifovski, V., Lohman, G., Pirahesh, H., Märschel, M.: ROX: Relational Over XML. In: (e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, pp. 264–275 (2004)
5. Shanmugasundaram, J., Kiernan, J., Shekita, E., Fan, C., Funderburk, J.: Querying XML Views of Relational Data. In: VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, Roma, Italy, pp. 261–270 (2001)
6. Shao, F., Novak, A., Shanmugasundaram, J.: Triggers over XML views of relational data. In: Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, Tokyo, Japan, pp. 483–484 (2005)
7. Agichtein, E., Josifovski, V.: Extracting Relations from XML Documents. In: Conceptual Modeling for Novel Application Domains, ER 2003 Workshops, Chicago, IL, USA, pp. 390–401 (2003)
8. Bohannon, P., Freire, J., Roy, P., Simeon, J.: From XML Schema to Relations: A Cost-Based Approach to XML Storage. In: Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, p. 64 (2002)
9. Tatarinov, I., Viglas, S., Beyer, K., Shanmugasundaram, J., Shekita, E., Zhang, C.: Storing and querying ordered XML using a relational database system. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA, pp. 204–215 (2002)
10. Garofalakis, M., Gionis, A., Rastogi, R., Seshadri, S., Shim, K.: XTRACT: A System for Extracting Document Type Descriptors from XML Documents. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, Texas, USA, pp. 165–176 (2000)

Data Mining Project: A Critical Element in Teaching, Learning and Assessment of a Data Mining Module

Hongbo Du

Department of Applied Computing, University of Buckingham,
Buckingham MK18 1EG, United Kingdom
hongbo.du@buckingham.ac.uk

Abstract. Data mining has been introduced into computing curricula. A data mining module should emphasise not only the technical but also the practical sides of the subject. This paper stresses the importance of using a data mining project as a critical element of the coursework. The paper outlines the intended learning outcomes and the expectations from students. The paper proposes a framework for project administration and assessment. By using a number of past projects as case studies, the paper demonstrates the project work involved and summarises good and bad experiences in running the project. The paper highlights the uncertain nature of data mining and consequent challenges and difficulties. The paper is intended to contribute towards a wider debate over the best practices in teaching, learning and assessment of data mining.

Keywords: Data mining, module project, learning, teaching, assessment.

1 Introduction

Data mining is a popular and interesting subject in computing, and has started to appear in undergraduate and postgraduate computing curricula, either in the form of a full module or as a part of a module on business intelligence or advanced databases ([3], [7], [10]). Because of the diversity in student backgrounds and module intentions, different approaches and methods in teaching, learning and assessment have been practised.

Data mining involves not only theories and techniques of computation but also processes, tasks and trade-offs of discovery concerns. Learning the subject is not only about knowledge and understanding but also about experience and practical skills. This balance should be reflected in the intended learning outcomes, the teaching and learning strategies, and the assessment criteria for the module. The argument naturally leads to using a data mining project as a major component of the coursework. It is felt that a data mining project should play a critical role for a data mining module in the same way as a database design project does for a database module. Although Rob and Ellis briefly mentioned using two types of projects in data warehousing and data mining [9], using data mining project for teaching, particularly in the UK, is still rare.

Similar attempts by this author before 2000 ran into various difficulties due to lacks of useful software tools, relevant guidelines and good cases of reference. In recent years, however, the teaching environment for data mining has been greatly

improved. An increasing number of software tools have become available for tutors to select. In 2000, the Cross Industry Standard Process for Data Mining (CRISP-DM) was published [2]. For the first time, a rigorous step-by-step industrial standard methodology has been introduced to and endorsed by many data mining practitioners. The methodology provides students with a complete lifecycle to mimic and a guideline for detailed actions and tasks to follow. Furthermore, an increasing number of successful cases of data mining have been reported ([1], [5], [6]). These cases become good references for students in preparation for their own projects.

A data mining project is harder than a database design project due to the uncertain nature of data mining, and therefore faces its own difficulties and challenges. This paper is intended to share the author's experience in this regard. The paper is a follow-up of an early work regarding the design of a data mining module for an undergraduate computing programme [3]. A data mining project is intended as a major part of the coursework for that module.

The rest of the paper is organised as follows. Section 2 outlines a specification of a data mining mini-project. The paper then addresses related issues arising from the specification, and proposes a framework for administrating and assessing various aspects of the project. In section 3, the paper uses a number of selected projects from the author's own classes as case studies and measures their successes according to the proposed framework. In section 4, the paper highlights the uncertain natures of data mining as well as the challenges and the difficulties involved, and summarises some useful lessons learnt.

2 Data Mining Mini-project: A Specification

2.1 Project Aim, Objectives and Scope

The data mining mini-project, referred to as *the project* hereafter, is concerned with discovering possible hidden patterns from a given data set by using a data mining software tool. The purpose of the project is to provide students with an opportunity to experience the complete lifecycle of data mining. In particular, students are required to follow the principles of the CRISP-DM methodology, define and undertake relevant tasks, exercise judgement and make justifiable decisions over relevant issues throughout the whole data mining process.

The key word here is *experience*. The project makes the students go through the practical process and face real challenges of making decisions in uncertain situations. It is unrealistic, however, to treat the project as real-life data mining and expect students handling it as professionals. Data mining is an art that requires a lot of practice to master. Consequently, the usefulness of the discovered patterns is much less important than what the students learn through their experience. Again, an analogy to a database design project can be drawn: we are more interested in the process of developing a database than the final database product.

Because of the discovery nature of data mining, the project scope must be controlled carefully. First, the project should be a joint group work by 2 or 3 students taken over a period of 5 to 6 weeks. The group project enables sharing of workload and at the same time encourages debates over related issues. Second, the project

should normally involve *one* of three main types of data mining, i.e. classification, cluster analysis and association discovery. In the cases when more than one type is required for the intended business purpose, the number of mining tasks for each type should be limited. Doing one thing properly is always more desirable than attempting many superficially. Comparing to real-life data mining, this project is indeed a small scale mini-project in every way.

The project suits a full module in the final year (FHEQ level 6) worth 15 to 20 units of credit. A specification for the module and its pre-requisite are presented in [3]. Some elementary knowledge in probability and statistics is assumed.

2.2 Project Content and Deliverables

The project should concentrate on the following main stages of the data mining process:

1. Data understanding. This stage involves activities in studying the data and data backgrounds, understanding related business activities from which the data are collected, conducting exploratory summaries and outlining possible directions for discovery.
2. Data preparation. The project work at this stage includes tasks in preparing and formatting data, pre-processing the data (such as discretisation, transformation, attribute selection, sampling, etc.) and if possible improving data quality.
3. Data modelling/mining. This stage is concerned with selecting suitable data mining solutions, setting appropriate parameters for the solutions, observing results and deciding if alternative mining solutions are needed, and whether any further data preparation is required before another round of mining begins.
4. Post processing. This stage of the project involves collecting results, evaluating the patterns for their significance and quality, attempting to interpret the patterns, and evaluating their fitness to the purposes outlined in 1.

Two main phases of the CRISP-DM standard, i.e. business understanding and deployment, have not been mentioned. This is because the true and complete business context of a selected data set may not be available, and hence it is difficult to *mock* the business reality. However, the tutor and students should seek maximal amount of information about the data background from limited sources. Students should make effort in considering possible deployments of useful patterns given the limited understanding of the application.

The deliverables for the project include a written report and an oral presentation from each project group. The report documents details of the project work and rationales behind them at each stage of the data mining process. The oral presentation aims to outline main issues with the data set, highlight major project tasks and key findings, justify any decisions taken, and defend the project work.

2.3 Related Issues

A number of issues regarding the project must be addressed. First, a suitable data set should be located. Such data sets were hard to find in the early years. Since 2005, increasing numbers of data sets from the public domain and commercial sources have become available online [8]. The project may require a single data set for all project

groups, or different data sets for different project groups. Given the time constraint and the project complexity, only one data set should be used by a project group. It should not be too large in size or too high in dimensionality. A data set with hundreds or even thousands of records and tens of variables would be ideal.

One potential concern regarding data sets is permission for use. Most data sets from the public domain or downloaded from the internet come with such permission. Nonetheless, both students and the tutor must check if the permission has been granted before using a data set.

Data mining software is another issue to be addressed. Existing software can be categorised into commercial systems and free tools. The commercial systems, e.g. Oracle Data Miner, are built to cope with the workload of real-life data sets of large sizes and high dimensionality. However, these systems are often cumbersome to learn and use with limited choices of solutions. On contrast, the free tools are often light-weight, easy to learn and use. Although many free tools fail to cope with data sets of extremely large sizes and high dimensionality, they should be sufficient for the kind of data sets for the project. Weka [12] is a free downloadable tool that has been widely used. Its Explorer module has a simple graphical user interface through which small-scale data mining and data exploration can be performed. The Knowledge Flow module can be used for a more serious piece of data mining through carefully designed task flows. The Experimenter module enables comparison on performances of classification methods. The free license overcomes the availability constraint.

A single data mining tool may not always meet all requirements of the project. For example, some data pre-processing may be better done using another tool such as Microsoft Excel before the data set is loaded into Weka. All practical knowledge of the mining tool is acquired through purposely designed practical classes. Practical knowledge of other software tools can be obtained either via added practical sessions for the module or through transferable skills from early modules.

2.4 Administration of the Project

The administration of the project follows the data mining lifecycle as described in 2.2. At the beginning, the tutor provides students with a specification document and even a presentation. This is followed by a period for data selection and group formation by students. Each group has a leader. The role may be taken by a specific member or played in rotation by all members of a group. Each group should then arrange a start-off meeting with the tutor to gain more understanding about the background of the chosen data set and present a project plan. During the project period, each group should hold regular meetings with the tutor to report the project progress and discuss issues arising. The tutor should by no means intervene in project decisions and activities. The tutor should play the roles of a monitor, a critic and a fictional client. The tutor monitors student progress through a sequence of small deliverables such as verbal reports, demonstrations, etc. By the end of the project, the reports from all groups may be compiled into a single proceeding and shared among all students of the class before the oral presentation is held.

Ideally, the module progression on key topics of the subject should coincide with the project lifecycle. At the beginning when the project specification is given out, the module introduces data mining concepts, principles and methodologies. The module

then proceeds on topics about data exploration and pre-processing, followed by basic mining and modelling techniques and evaluation of patterns. The basic techniques are then followed by advanced and alternative techniques. The module covers application issues towards the end. This approach of module delivery enables the students to gain knowledge from classes and then apply it practically in the project.

2.5 Assessment Framework of the Project

The project is assessed according to the quality of work at each stage of the data mining process. Correctness, completeness and soundness of judgement are the main factors for determining marks. Table 1 presents a framework for project assessment. The matrix highlights the assessment focuses by each factor at each stage. Further details for assessment can be found in [2].

Table 1. Assessment Framework

Phases Factors	Data Understanding & Exploration (20%)	Data Preparation & Pre-processing (25%)	Data Modelling & Mining of Patterns (25%)	Evaluation of Result Patterns (20%)
Correctness	Correct understanding of data characteristics and features	Correct data pre-processing and preparation operations	Correct mining tasks, sensible choices of modelling solutions. Sensible setting of parameters.	Correct understanding of evaluation metrics and interpretation of patterns
Completeness	Coverage of aspects of data features such as data types, distributions, missing values, etc.	Sufficiency of the operations for the purpose of discovery	Using alternative solutions. Alternative setting of parameters. Comparison of solutions.	Complete collection, summarisation and categorisation of patterns. Evaluation of both pros and cons.
Soundness of Judgement	Needs for data pre-processing	Justification for the operations and their relevance to mining	Justification for selection of mining operations and parameter settings	Need for further mining. Selection/Identification of interesting patterns.
Project Planning, Execution, Management and Teamwork (10%)				

Given the importance of all the stages, marks should be evenly divided. Because of the complexity and amount of time required, the Data Preparation and Pre-processing stage and the Data Modelling/Mining stage may take a marginally larger share of the total mark than the other stages. A certain small percentage of the total mark may be given to the successful planning, effective execution and management of the project,

and collaborative teamwork. The distribution of percentage marks under the stage headings in Table 1 is an example scheme practised by this author.

According to the framework, a project can be broadly classified into one of the following four categories:

1. Unsatisfactory. This kind of projects normally has major flaws in project activities at certain stage(s). Without a clear discovery aim, random decisions are made and random actions are taken. Data are not examined and well prepared before mining. Certain solutions with default parameter settings are chosen without any good reasons. Irrelevant patterns with little interpretation are collected as result. Little attention is paid to evaluation of the result patterns. The poor quality of work reflects no serious attempt. The total mark awarded for this category should be lower than the bare pass mark (e.g.40%).
2. Fair. This kind of projects normally produces some positive results in terms of experience. However, the work is not well planned. It involves either too many trial-and-error or limited project activities. Students show limited understanding of knowledge and make decisions without full consideration of the issues concerned. Some directly “copy-and-paste” style references to seen examples are made without questioning the relevance. Limited understanding about evaluation of result patterns is evident. The total percentage mark for this category is between the pass and a lower 2.II (e.g. between 40% and 54%).
3. Good. This kind of projects shows sufficient understanding and good application of knowledge. The objectives are clear. Project activities are planned and targeted not randomly decided. Actions taken in preparing data are well thought with good supporting arguments. The selection of mining methods is sensible, and so are settings of relevant parameters. Alternative methods or alternative parameter settings are tried with justification. There are clear evidences of appreciation of evaluation of result patterns. Sensible interpretations and implications for course of actions are drawn from the mining results. The total percentage mark for this category is between a higher 2.II and a high 2.I (e.g. 55% and 69%).
4. Excellent. This kind of projects shows all the merits of the category above and furthermore demonstrates excellent performance throughout the entire process. A sense of critical analysis and critical evaluation is demonstrated at every stage. There is a well-thought reasoning from a business objective to data mining tasks. All decisions and actions in data preparation and pre-processing are supported by sensible arguments. Data mining tasks are well defined and relevant to the aim. Each trial of data mining serves a clear purpose. The evaluation of resulting patterns is thorough and appropriate, and influences the selection of useful patterns for potential use. The total percentage mark for this category is a clear first (e.g. $\geq 70\%$).

In practice, the marking can be done in a top-down or a bottom-up fashion. In the bottom-up approach, detailed percentage marks are first given to each stage of the project according to the factors outlined in the assessment framework. The total mark for the whole project should then reflect the appropriate category for the project. In the top-down approach, the assessor first classifies the project into one of the categories according to the category descriptors, and then fine tune the percentage marks for each stage to reflect the description of the category.

The report and the oral presentation should not be marked separately. It is suggested that a provisional mark is first given to the work described in the report. This mark is either confirmed or adjusted accordingly at the oral presentation.

3 Case Studies: The Good, the Bad and the Ugly

The project was first used in the author's data mining module in 2001. The project has become an established part of the coursework ever since. The weight for the project in the coursework has increased from a mere 40% at the beginning to 60% later and to the current 100% in 2010. In 2007, the author created a data mining module at Sarajevo School of Science and Technology. The project was and still is a significant part (60%) of the coursework for that module.

Between 2001 and 2005, the author mainly relied on data sets from the UCI Machine Learning Repository [11] and a few real-life data sets from his consultancy work. Since then, data source was no longer a major concern. Before 2006, IBM Intelligent Miner for Data was used as the data mining tool. Since 2006, Weka has been used for its rich solutions, simplicity and availability for students. Microsoft Excel has also been used for assisting data exploration and data pre-processing.

Figure 1 shows the distribution of the project works in the four categories for the 35 project groups collected between 2002 and 2010 for the purpose of this paper. It is true that most of the classes are fairly small in size, and the biggest class size is 34 students (11 groups). Majority of project results fall into Fair and Good categories while Unsatisfactory and Excellent projects are in the minority. This seemingly good result reflects the practice that we mark on the project experience and what students learn from it. This paper selects three typical projects from the 35 group projects as case studies. The purpose is to show the nature of the tasks involved, student works produced and assessment of their works.

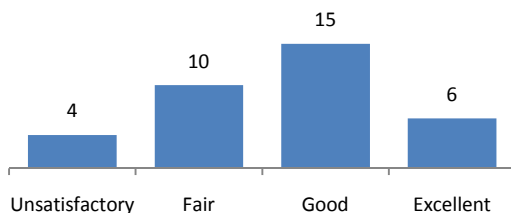
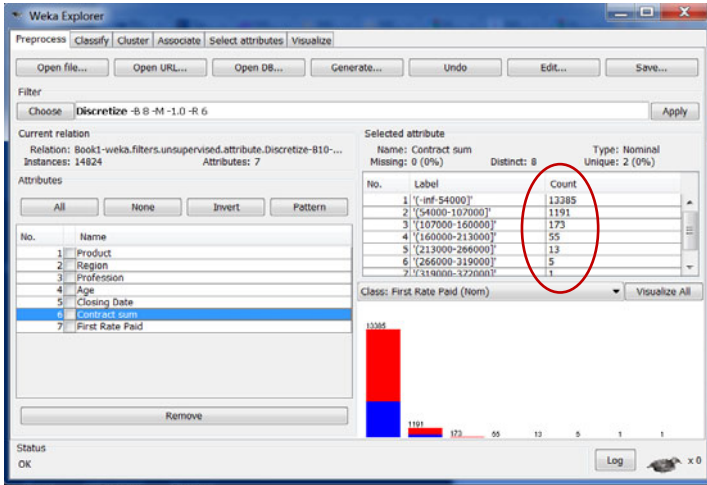


Fig. 1. Distribution of Project Categories

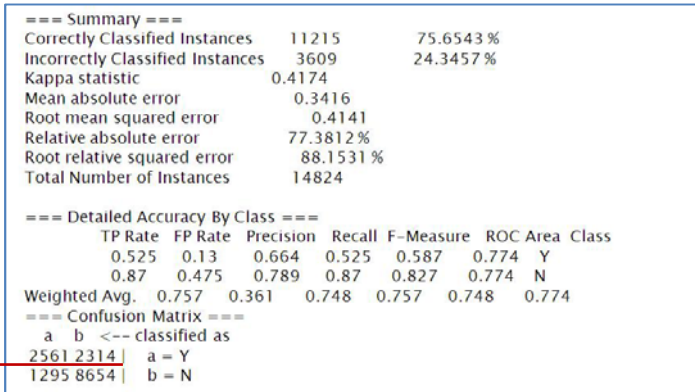
3.1 Project One: The Bad

The Data Set

The data set for this project is about insurance purchases. It has 14,845 recordings about purchased product type, regional code, profession, age, first rate paid, contracting sum and closing date. Potentially interesting patterns include a classification model regarding which type of product is purchased by what kinds of customers.



(a)



(b)

Fig. 2. Some Results from Project One

The Student Work

The project was undertaken by a group of two students in 2010. Details of their work are listed as follows:

- Data understanding. Besides known facts about the attribute domains and the data set size, virtually nothing more on data understanding was done. Some anomaly records (farmers of age 3 and car insurance buyers under the legal minimum age for driving) were spotted. No discovery objectives were mentioned.
- Data preparation and pre-processing. Regional codes were replaced with nominal labels (A, B, C, D). The unknown regional code 999 was converted to the unknown symbol recognisable by Weka. The anomaly records were removed from the data set. The justification given was that 21 anomalies count only 0.14% of the total number of records. The values for age attribute were discretized using an

unsupervised equal-length method with default parameter setting (10 bins). The values for the contracting sum attribute were also discretized into 8 bins using the same method. Figure 2(a) shows the result of the discretisation for the contracting sum attribute.

- Data modelling/mining. One classification model using a decision tree method was obtained. The tree has an overall accuracy of 75%. Figure 2(b) presents the evaluation details of the tree. 8 quantitative association rules with support of 10% and confidence of 91% were also discovered. No explanation was given about the selection of the rules. Redundancy exists between two of the rules.
- Post-processing. Little attempt regarding evaluation of patterns was made. No clear interpretation of the patterns was given.

The Assessment

The project does not outline any directions for the discovery. Understanding of data characteristics is limited, which leads to the random decision of using an unsupervised equal-length method for data discretisation. No reasons were given regarding why 10 and 8 bins are chosen for discretising the age and contracting sum attributes. A clear sign of concern, as indicated by the circle in figure 2(a), was ignored. Some credits should be given for the handling of the anomalies and the replacement of regional codes. The project shows serious weaknesses in the data modelling/mining stage. Only one trial of decision tree induction was attempted without justification. The purpose of the association rules is not clear. The weakest point of the whole project is post-processing. Little attention was paid to the performances of the patterns. The students did not realise that the tree is almost useless for classifying who are paying for the first rate (as indicated by the the confusion matrix in figure 2(b)). The breakdown of marks is as follows: 5 out of 20 to Data Understanding, 12 out of 25 to Data Preparation and Pre-processing, 10 out of 25 to Modelling/Mining, and 4 out of 20 to Post-processing. Because of the disorganised approach to work, only 3 out of 10 marks were given to the project management. With the total mark of 34%, the project is unsatisfactory.

3.2 Project Two: The Good

The Data Set

This project uses a public domain data set about heart diseases donated by Cleveland Clinic Foundation. The data set has 303 records and 14 attributes. The attributes represent patient age, patient gender, and a range of clinic test results. The result measurements include chest pain type, resting blood pressure, amount of cholesterol, fasting blood sugar greater than 120, resting electrocardiographic result, maximum heart rate, presence of exercise-induced angina, ST depression, slope of peak exercise ST segment, number of major vessels coloured by fluoroscopy, thalassaemia, and angiographic disease status. The number of data records is limited. Students are expected to show good use of the limited data. Potentially interesting patterns would be classification models regarding the presence of heart disease.

The Student Work

The project was undertaken by a group of three students in 2007. Details of the work are listed as follows:

- **Data understanding.** From the start, the group outlined a clear business objective, i.e. finding patterns relating to the presence or absence of the heart disease. The group conducted operations such as collecting and formatting data, exploring domain types and values, obtaining descriptive statistics for numerical attributes, and assessing data quality. Separate reports for the purposes were also produced.
- **Data preparation and pre-processing.** The group focused on data cleaning by removing outliers and filling missing values with sensible alternatives. For both purposes, the ordinal and nominal values were first converted into discrete integers. To deal with missing values, the group decided to find the record's nearest neighbour and use the attribute value of the neighbour to fill the missing field. To deal with outliers, the students first plotted the data records as points in a scatter plot and manually located those anomaly values. An anomaly value was considered as being wrongly entered and hence also replaced by the value of its nearest neighbour. After the data cleaning operations, the discrete integers for ordinal and nominal attributes were converted back to the original labels.
- **Data modelling/mining.** The group conducted two main data mining tasks: to build a model to classify if a patient is healthy or having the disease, and to profile patients in both classes via clustering. For the first task, the group used J4.8 decision tree method with different parameter settings and 2/3-1/3 split of training-testing examples as the test option. A number of possible trees with overall accuracy rates from 72% to 79% were obtained. The students realised that pruning improves the tree accuracy. Figure 3 shows the performance summary of one of the candidate trees. To consolidate the finding, a similar classification task was also attempted by using the tree induction method of another tool (RDS). Some similarities in the resulting trees were found. For the second task, the k-means method was used with tweaking of different k values for good cluster quality, and eventually the optimal value for k was set to 4.
- **Post-processing.** Both the decision trees and the clusters were evaluated when different parameter settings for the tree induction and different values for k were attempted. The group also converted the trees into rules to assist their understanding. By consulting external medical experts, some of the rules made good medical sense, and supported recommendations for certain people to avoid having heart disease. The interpretation of clustering results was attempted via cluster summary and through visualising membership of the clusters.

The Assessment

This project has paid sufficient attention to every task at every stage of the data mining process. The project adheres to the CRISP-DM guideline and the tasks are performed in a systematic manner. The business objective of discovery is outlined and related to the data mining goals. Data characteristics are studied carefully, but data summary is done only for numeric data. The methods for cleaning the data are

sensible and can be justified. In order to perform data mining tasks thoroughly, the group decided to limit data mining to classification and clustering only, which is sensible. The project has shown repeated attempts for both mining tasks in order to obtain the best results. Such controlled trial-and-error activities are appropriate. However, the group did not attempt alternative classification techniques, and did not conduct a comparative study among the alternative approaches. It is also questionable whether using a 2/3-1/3 split is the most effective use of the limited data. The group considered evaluation seriously and used the evaluation results to determine a better model. However, the group did not give the presence of the heart disease higher priority and look for models with better true positive performance. Consequently, 15 marks were given to Data Understanding, 18 to Data Preparation and Pre-processing, 15 to Data Modelling/Mining, and 14 to Post-processing. Because of the good organisation and documentation of work, 7 out of 10 marks were given to the project management. With the total mark of 69%, the project is standing at the border between good and excellent. The project did not get a clear first due to the limitations in modelling/mining and evaluation.

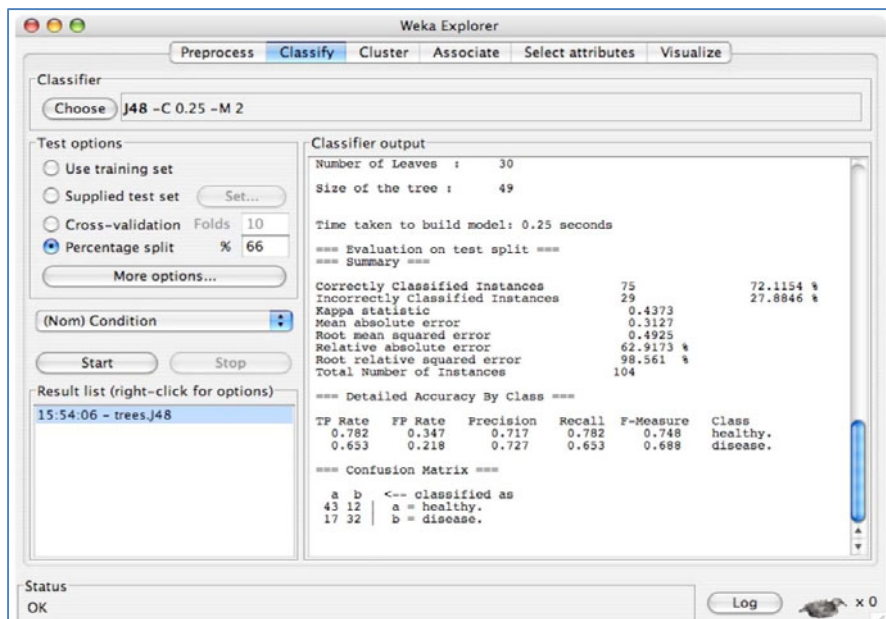


Fig. 3. Selected Results from Project Two

3.3 Project Three: The Ugly

The Data Set

The data set used for this project is the same insurance data set used for project one.

The Student Work

The project was undertaken also in 2010, but by a single student against the tutor's advice. Key points of the project work are summarised as follows:

- **Data understanding.** The student decided to carry out a brute force bottom-up discovery of any potential patterns. At this stage, the student used Weka and Excel to gain understanding about domain types, value distributions of attributes, and unknown values. Unlike the group for project one, this student did not identify any anomalies, but spotted that values for the contracting sum attribute were extremely skewed towards the lower end.
- **Data preparation and pre-processing.** Similar to project one, regional codes were replaced with nominal labels (A, B, C, D and E, where E for unknown). The age attribute was discretized into more natural age groups such as child, teenager, young, adult and senior. Because of the skew of the contracting sum values, the student decided to apply logarithm transformation on the original values so that the levels of magnitude instead of the actual figures of contracting sums were considered.
- **Data modelling/mining.** The student took the decision to do every data mining task: classification, clustering and association mining. The student laboriously tried 4 methods for clustering, 10 methods for classification, and 2 methods for association rule discovery. For clustering, different values of k were attempted for the k -means and the EM methods. For classification, the student attempted to induce classification models for product types, and used the Weka Experimenter to compare performances among the classification methods. Little explanation was given regarding the setting of parameters. For association rule discovery, confidence and accuracy were used for selecting top 10 rules.
- **Post-processing.** The student was conscious about the value for k and used the evaluation of cluster quality to determine the optimal value. However, except the performance analysis using Experimenter, very little attention was paid to the detailed performance evaluation of different classification models shown in confusion matrices. The student did not notice the strength of JRip method in classifying life insurance buyers. The student did not pay attention to the meaning of association rules at all.

The Assessment

The project is a showcase of trial-and-error gone to the extreme. Many trials were made and many patterns were discovered, but these patterns are not carefully examined. There are gems of good ideas here and there in some individual tasks, such as the logarithm transformation for the contracting sum attribute, the comparative study of techniques for classification, etc., but the project as a whole is not a piece of coherent work. The student did not realise the complexity of most decision trees, and totally ignore the inappropriate associations (e.g. contracting sums with their logarithm-transferred values). The project can only be classified as fair with a total percentage mark of 48% (10 for Data Understanding, 15 for Data Preparation and Pre-processing, 10 for Data Modelling/Minning, 8 for Post-processing and 5 for project management).

4 Discussions

Uncertainty and Difficulty

Uncertainty is the most noticeable feature of data mining. Given the same data set and the same objectives, there can be various ways of preparing and mining the data. Because of the uncertainty, trials of alternatives are unavoidable. The exercise of sound judgement is therefore essential. The sense of sound judgement depends on levels of understanding of knowledge, analytic skills and experience. This means that data mining projects can be seen as time consuming and difficult. However, the difficulty should not come as a surprise, nor should it suggest that the project is unsuitable for undergraduate students. Data mining should be taught to final year undergraduate students upon whom a greater degree of academic maturity in terms of analytic skills, logical reasoning and soundness of judgement is expected. With advices and directions from the tutor, the difficulty can be overcome.

There is a genuine difficulty, i.e. the absence of domain experts from the data mining lifecycle. *Domain experts* are those who know the application domain well and can judge which patterns are potentially useful and which are not. The experts are in fact present throughout the data mining process in most, if not all, real-life data mining projects [4]. It is not realistic to expect the tutor to play such a role for various domains of application for all kinds of data sets. This particular difficulty could be avoided by the tutor intervening in the data set selection and only allowing students to select a data set where the tutor is familiar with the application domain. By doing so, however, the motivation of the students may be affected.

Usefulness of Case Studies

Because of the uncertainty and difficulties, it is useful and beneficial to study cases of reported data mining projects. These case studies bring the data mining process alive and provide students an opportunity to observe how data mining is done before they try it themselves. The value of case studies cannot be underestimated. Good case studies were rare and difficult to get. Indeed, industry and commerce often consider data mining as a closely guarded secret. However, the situation appears changing. It is now possible to find successful commercial data mining projects ([4], [5]). It is recommended that a successful case be presented in teaching sessions so that every aspect of the case can be discussed thoroughly under the tutor's supervision.

The author has used an assignment as a way of getting good case studies: students are asked to search for a good case in a designated application area from the published sources. They are then required to study the case, present it and comment on it. This work has been proved very useful for students, and can be taken as a part of assessment. Such an assignment may also be considered as a part of the coursework element for modules that only briefly cover the topic of data mining.

Levels of Expectation and Project Scope

The level of expectation must reflect the aim of the project, i.e. to provide students with an opportunity to *experience* data mining. It is necessary to emphasise again that the success or failure of a project should not be judged on applicability of the result patterns, but on the tasks and actions taken and their justifications.

The amount of work must justify the amount of time allocated. The project specification given in section 2 is meant for a full data mining module on a computing programme. The scope of the project must be adjusted accordingly if the module has a different emphasis or it is for a different programme. For instance, a module on advanced databases with a significant part on data mining may require a much smaller scope. Everything within the project, such as the data set dimensionality, the amount of data preparation work and the number of data mining tasks, should all be limited. The project group size may even be increased to 4 students rather than 2 or 3 to further spread workload.

Student Feedbacks

A thorough survey of student feedbacks on the project over years has not been conducted yet mainly because of small class sizes. However, from the annual student feedback questionnaires on the module in which a question about the coursework is asked, comments on the use of the project are overall positive. Most students consider the project experience useful, and it has helped enhancing the understanding of the subject. At the same time, many students consider the project harder than practical projects they have done for other subjects. Weka as a tool for supporting the project has also received good feedback for its simplicity and ease of use.

Some cautious feedback comments are also received. Due to the lack of business context and short of domain experts, some students cannot see how useful result patterns can be, which in turn affect the level of their confidence towards the patterns they have discovered. The biggest difficulty is how to deal with the uncertainty. This could be a cultural shock for computing students who have worked largely towards deterministic solutions. Some students also have difficulties in data understanding and pattern evaluation due to lack of training and understanding of basic statistics.

A small minority of students have very little clues about the project. They tend to make bad judgement and rely heavily on trial and error or hand-on guidance from the tutor. Their comments towards the project are quite negative. Effort is certainly needed to find out how to minimise the size of this group.

Resource Implications

Running the project may require additional resources in terms of tutor's hours. Enhancing student learning experience has cost. Indeed, no pain, no gain. In the author's experience, however, the extra hours spent on administrating/supervising the project is only marginally more than that for a database design project, thanks to the academic maturity of the final-year students. The good students usually need very little guidance. They can obtain the required knowledge from lectures and reference materials. Weak and fair students in fact take up most of the consultation time. Whether the project idea can scale up to classes of hundreds of students is yet to be verified.

5 Concluding Remarks

This paper argues that a data mining mini-project should become a critical part of the coursework for a data mining module. The project should benefit student learning experience of this interesting subject in the same way as a database design project

does for databases. Data mining may be more challenging to do for students and harder to manage for the tutor than a database design project, and hence might require more resources in terms of tutor's hours. Despite all kinds of difficulties, the paper argues and demonstrates that such a project should be feasible for final year students of computing major. With an ambition to practise data mining, the expectation must be realistic: it is more about experience than discovery results.

The assessment framework proposed in this paper has been practised and refined by this author. It undoubtedly needs the scrutiny of the teaching community. It is hoped that the framework will be continuously refined to improve the administration and assessment of such an important element.

Acknowledgement. The author wishes to thank all the students for using their works as case studies in this paper. For obvious reasons, their names are kept anonymous.

References

1. Berry, M.J.A., Linoff, G.: *Mastering Data Mining: the Art and Science of Customer Relationship Management*. John Wiley & Sons, Chichester (2000)
2. Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., Wirth, R.: *CRISP-DM 1.0: Step-by-Step Data Mining Guide*, SPSS (2000)
3. Du, H.: Teaching Undergraduate Students Data Mining: Ideas, Experience and Challenges. In: 8th International Workshop in Teaching, Learning and Assessment of Databases (TLAD), pp. 49–54. University of Abertay Dundee (2010)
4. Du, H.: *Data Mining Techniques and Applications, An Introduction*, Cengage Learning: Andover (2010)
5. Kitts, B., Melli, G., Rexer, K. (eds.): *Data Mining Case Studies*, In: *The First International Workshop on Data Mining Case Studies*, 2005 IEEE International Conference on Data Mining, Huston, USA (2005)
6. Luan, J., Zhao, C.-M. (eds.): *Data Mining in Action: Case Studies of Enrolment Management*. Wiley Periodicals Inc., Chichester (2006)
7. Mrdalj, S.: Teaching An Applied Business Intelligence Course, *Issues in Information Systems*. *Issues in Information Systems VIII(1)*, 134–138 (2007)
8. Piatetsky-Shapiro, G.: <http://www.kdnuggets.com/> (accessed March 30, 2011)
9. Rob, M.A., Ellis, M.E.: Case Projects in Data Warehousing and Data Mining. *Issues in Information Systems VIII(1)* (2007)
10. The Quality Assurance Agency for Higher Education: *Subject Benchmark Statements, Computing* (2007)
11. The University of California at Irvine: *UCI Machine Learning Repository*, <http://archive.ics.uci.edu/ml/about.html> (assessed on March 30, 2011)
12. Witten, I.H., Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd edn. Morgan Kaufmann Publishers, San Francisco (2011)

Author Index

- Aberer, Karl 2
Adaikkalavan, Raman 68
Ali, Muhammad Intizar 132
- Bätz, Alexander 154
Bell, David A. 140
Bi, Yaxin 83
Bizer, Christian 1
Böhlen, Michael 128
Bokermann, Dennis 209
Böttcher, Stefan 154, 182, 209
Brenninkmeijer, Christian Y.A. 136
- Cavaliere, Federico 167
Chakravarthy, Sharma 68
Conroy, Kenneth 97
- Deshpande, Nikhil 68
Dittmann, Jana 3
Dobbie, Gillian 33
Du, Hongbo 221
Dustdar, Schahram 132
- Fernandes, Alvaro A.A. 136
- Galpin, Ixent 136
Glavic, Boris 128
Gray, Alasdair J.G. 136
Guerrini, Giovanna 167
- Hartel, Rita 154, 182, 209
Hedeler, Cornelia 108
Hong, Jun 140
- Kanne, Carl-Christian 128
- May, Gregory C. 97
Merkel, Ronny 3
Mesiti, Marco 167
Mukherji, Abhishek 48
Muzammal, Muhammad 118
- Naeem, Muhammad Asif 33
- Paton, Norman W. 108, 136
Pichler, Reinhard 132
- Rahman, Syed Saif ur 18
Roantree, Mark 97
Rundensteiner, Elke A. 48
- Saake, Gunter 3, 18
Schaefer, Gerald 66
Schäler, Martin 3
Schallehn, Eike 18
Schulze, Sandro 3
Stey, Sebastian 182
Suzuki, Nobutaka 194
- Taylor, Robert 136
Tilgner, Christian 128
Truong, Hong-Linh 132
- Ward, Matthew O. 48
Warrington, Giles 97
Weber, Gerald 33
Weng, Daiyue 140
Wu, Shengli 83
- Zeng, Xiaoqin 83