

**András Benczúr
Bernhard Thalheim
Tomáš Horváth (Eds.)**

LNCS 11019

Advances in Databases and Information Systems

**22nd European Conference, ADBIS 2018
Budapest, Hungary, September 2–5, 2018
Proceedings**

 **Springer**

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, Lancaster, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Zurich, Switzerland

John C. Mitchell

Stanford University, Stanford, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

C. Pandu Rangan

Indian Institute of Technology Madras, Chennai, India

Bernhard Steffen

TU Dortmund University, Dortmund, Germany

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max Planck Institute for Informatics, Saarbrücken, Germany

More information about this series at <http://www.springer.com/series/7409>

András Benczúr · Bernhard Thalheim
Tomáš Horváth (Eds.)

Advances in Databases and Information Systems

22nd European Conference, ADBIS 2018
Budapest, Hungary, September 2–5, 2018
Proceedings

Editors

András Benczúr 
Eötvös Loránd University
Budapest
Hungary

Tomáš Horváth
Eötvös Loránd University
Budapest
Hungary

Bernhard Thalheim 
Christian-Albrechts-Universität
Kiel
Germany

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-98397-4 ISBN 978-3-319-98398-1 (eBook)
<https://doi.org/10.1007/978-3-319-98398-1>

Library of Congress Control Number: 2018950525

LNCS Sublibrary: SL3 – Information Systems and Applications, incl. Internet/Web, and HCI

© Springer Nature Switzerland AG 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, 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.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Leonid Andreevich Kalinichenko

10 June 1937 – 17 July 2018



One of the pioneers of the theory of databases passed away. The name of Leonid Andreevich is rightly associated with the development of promising directions in databases, the founding of an influential scientific school, the creation of two regularly held international scientific conferences. Leonid Andreevich Kalinichenko received his Ph.D. degree from the Institute of Cybernetics, Kiev, Ukraine, in 1968 and his degree of Doctor of Sciences from the Moscow State University in 1985. Both degrees in Computer Science.

He served as Head of the Laboratory for Compositional Information Systems Development Methods at the Institute of Informatics Problems of the Russian Academy of Science, Moscow. As Professor, he taught at the Moscow State University (Computer Science department) courses on distributed object technologies and object-oriented databases. His research interests included: interoperable heterogeneous information resource mediation, heterogeneous information resource integration, semantic interoperability, compositional development of information systems, middleware architectures, digital libraries.

His pioneering work on database model transformation (VLDB 1978) and fundamental book on data integration “Methods and tools for heterogeneous database integration” (Moscow, 1983) were ahead of time. His theoretical studies of semantic interoperability attracted attention of the international community in the 90s, when interoperability became a hot topic. He is also a co-author of the four books: “SLANG - programming system for discrete event system simulation” (Kiev, 1969), “Computers with advanced interpretation systems” (Kiev, 1970), “Computer networks” (Moscow, 1977), “Database and knowledge base machines” (Moscow, 1990). He had a number of papers in journals and conference proceedings and served as PC member for numerous international conferences.

During the last few years his activities and work were devoted to problems in data intensive domains. In 2013–2016 he initiated several research projects aiming at conceptual modeling and data integration within distributed computational infrastructures. He also launched a Master program “Big data: infrastructures and methods for problem solving” at Lomonosov Moscow State University.

In addition to his own research, Leonid spent significant energy on integration of database research communities in different countries. He had a key role in the organization of a series of East-West workshops (Klagenfurt and Moscow), as well as of the ADBIS series of international workshops held in Moscow in 1993-1996. This series of workshops was transformed into ADBIS (Advances in Data Bases and Information Systems) conference series. He also established the “Russian Conference on Digital Libraries” (RCDL) in 1999, transformed into “Data Analytics and Management in Data Intensive Domains” conference (DAMDID/RCDL) in 2015. Finally, he launched the Moscow Chapter of ACM SIGMOD in 1992. Monthly seminars of this chapter play significant role in shaping local research and professional communities in Russia.

L.A. Kalinichenko was a member of the ACM, the Chair of the Moscow ACM SIGMOD Chapter, the Chair of the Steering Committee of the European Conference “Advances in Databases and Information Systems” (ADBIS), the Chair of the Steering Committee of the Russian Conference on Digital Libraries (RCDL), a Member of the Editorial Board of the International Journal “Distributed and Parallel Databases”, Kluwer Academic Publishers. All of us will remember him for what he did to build a wider community in Europe overcoming the divisions that had existed for decades. We, the members of the Steering Committee, remember him as a very generous person and an excellent scientist.

Paolo Atzeni
 Ladjel Bellatreche
 Andras Benczur
 Maria Bielikova
 Albertas Caplinskis
 Barbara Catania
 Johann Eder
 Janis Grundspenkis
 Hele-Mai Haav
 Theo Haerder
 Mirjana Ivanovic
 Hannu Jaakkola
 Marite Kirikova
 Mikhail Kogalovsky
 Margita Kon-Popovska
 Yannis Manolopoulos

Rainer Manthey
Manuk Manukyan
Joris Mihaeli
Tadeusz Morzy
Pavol Navrat
Mykola Nikitchenko
Boris Novikov
Jaroslav Pokorny
Boris Rachev
Bernhard Thalheim
Gottfried Vossen
Tatjana Welzer
Viacheslav Wolfengagen
Robert Wrembel
Ester Zumpano

Preface

The 22nd East-European Conference on Advances in Databases and Information Systems (ADBIS 2018) took place in Budapest, Hungary, during September 2–5, 2018. The ADBIS series of conferences aims at providing a forum for the dissemination of research accomplishments and at promoting interaction and collaboration between the database and information systems research communities from Central and East European countries and the rest of the world. The ADBIS conferences provide an international platform for the presentation of research on database theory, development of advanced DBMS technologies, and their advanced applications. As such, ADBIS has created a tradition with editions held in St. Petersburg (1997), Poznań (1998), Maribor (1999), Prague (2000), Vilnius (2001), Bratislava (2002), Dresden (2003), Budapest (2004), Tallinn (2005), Thessaloniki (2006), Varna (2007), Pori (2008), Riga (2009), Novi Sad (2010), Vienna (2011), Poznań (2012), Genova (2013), Ohrid (2014), Poitiers (2015), Prague (2016), and Nicosia (2017). The conferences are initiated and supervised by an international Steering Committee consisting of representatives from Armenia, Austria, Bulgaria, Czech Republic, Cyprus, Estonia, Finland, France, Germany, Greece, Hungary, Israel, Italy, Latvia, Lithuania, FYR Macedonia, Poland, Russia, Serbia, Slovakia, Slovenia, and the Ukraine.

The program of ADBIS 2018 included keynotes, research papers, thematic workshops, and a doctoral consortium. The conference attracted 69 paper submissions from 46 countries from all continents. After rigorous reviewing by the Program Committee (102 reviewers and 14 subreviewers from 28 countries in the PC), the 17 papers included in this LNCS proceedings volume were accepted as full contributions, making an acceptance rate of 25% for full papers and 41% in common. As a token of the appreciation of the longstanding, successful cooperation with ADBIS, Springer sponsored for ADBIS 2018 a best paper award. Furthermore, the Program Committee selected 11 more papers as short contributions. Authors of ADBIS papers come from 19 countries. The six workshop organizations acted on their own and accepted 24 papers for the AI*QA, BIGPMED, CSACDB, M2U, BigDataMAPS, and Current Trends in contemporary Information Systems and their Architectures (less than 43% acceptance rate of each workshop) workshops and three from the Doctoral Consortium. Short papers, workshop papers, and a summary of contributions from ADBIS 2018 workshops are published in a companion volume entitled *New Trends in Databases and Information Systems* in the Springer series *Communications in Computer and Information Science*. All papers were evaluated by at least three reviewers. The selected papers span a wide spectrum of topics in databases and related technologies, tackling challenging problems and presenting inventive and efficient solutions. In this volume, these papers are organized according to the seven sessions: (1) Information Extraction and Integration, (2) Data Mining and Knowledge Discovery, (3) Indexing, Query Processing, and Optimization, (4) Data Quality and Data Cleansing, (5) Distributed

Data Platforms, Including Cloud Data Systems, Key-Value Stores, and Big Data Systems, (6) Streaming Data Analysis, (7) Web, XML and Semi-structured Databases.

For this edition of ADBIS 2018, we had three keynote talks: the first was by Alexander S. Szalay from John Hopkins University, USA, on “Database-centric Scientific Computing,” the second by Volker Markl on “Mosaics in Big Data,” and the third by Peter Z. Revesz from the University of Nebraska-Lincoln, USA, on “Spatio-Temporal Data Mining of Major European River and Mountain Names Reveals Their Near Eastern and African Origins.”

The best papers of the main conference and workshops were invited to be submitted to special issues of the following journals: *Information Systems* and *Informatica*. We would like to express our gratitude to every individual who contributed to the success of ADBIS 2018. Firstly, we thank all authors for submitting their research paper to the conference. However, we are also indebted to the members of the community who offered their precious time and expertise in performing various roles ranging from organizational to reviewing roles — their efforts, energy, and degree of professionalism deserve the highest commendations. Special thanks to the Program Committee members and the external reviewers for their support in evaluating the papers submitted to ADBIS 2018, ensuring the quality of the scientific program. Thanks also to all the colleagues, secretaries, and engineers involved in the conference and workshops organization, particularly Altagra Business Services and Travel Agency Ltd. for the endless help and support. A special thank you goes to the members of the Steering Committee, and in particular its chair, Leonid Kalinichenko, and his co-chair, Yannis Manolopoulos, for all their help and guidance. Finally, we thank Springer for publishing the proceedings containing invited and research papers in the LNCS series. The Program Committee work relied on EasyChair, and we thank its development team for creating and maintaining the platform; it offered great support throughout the different phases of the reviewing process. The conference would not have been possible without our supporters and sponsors: Faculty of Informatics of the Eötvös Loránd University, Pázmány-Eötvös Foundation, and ACM Hungarian Chapter.



July 2018

András Benczúr
Bernhard Thalheim
Tomáš Horváth

Organization

Program Committee

Bernd Amann	Sorbonne University, France
Birger Andersson	Royal Institute of Technology, Sweden
Andreas Behrend	University of Bonn, Germany
Ladjel Bellatreche	LIAS/ENSMA, France
András Benczúr	Eötvös Loránd University, Hungary
András A. Benczúr	Institute for Computer Science and Control Hungarian Academy of Sciences (MTA SZTAKI), Hungary
Maria Bielikova	Slovak University of Technology in Bratislava, Slovakia
Zoran Bosnic	University of Ljubljana, Slovenia
Doukifli Boukraa	Université de Jijel, Algeria
Drazen Brdjanin	University of Banja Luka, Bosnia and Herzegovina
Albertas Caplinskas	Vilnius University, Lithuania
Barbara Catania	DIBRIS University of Genova, Italy
Ajantha Dahanayake	Lappeenranta University of Technology, Finland
Christos Doulkeridis	University of Piraeus, Greece
Antje Düsterhöft-Raab	Hochschule Wismar, University of Applied Science, Germany
Johann Eder	Alpen Adria Universität Klagenfurt, Austria
Erki Eessaar	Tallinn University of Technology, Estonia
Markus Endres	University of Augsburg, Germany
Werner Esswein	TU Dresden, Germany
Georgios Evangelidis	University of Macedonia, Thessaloniki, Greece
Flavio Ferrarotti	Software Competence Centre Hagenberg, Austria
Peter Forbrig	University of Rostock, Germany
Flavius Frasincar	Erasmus University Rotterdam, The Netherlands
Jan Genci	Technical University of Kosice, Slovakia
Jānis Grabis	Riga Technical University, Latvia
Gunter Graefe	HTW Dresden, Germany
Francesco Guerra	Università di Modena e Reggio Emilia, Italy
Giancarlo Guizzardi	Federal University of Espirito Santo, Brazil
Peter Gursky	P. J. Safarik University, Slovakia
Hele-Mai Haav	Institute of Cybernetics at Tallinn University of Technology, Estonia
Theo Härder	TU Kaiserslautern, Germany
Tomáš Horváth	Eötvös Loránd University, Hungary
Ekaterini Ioannou	Technical University of Crete, Greece
Márton Ispány	University of Debrecen, Hungary
Mirjana Ivanovic	University of Novi Sad, Serbia

Hannu Jaakkola	Tampere University of Technology, Finland
Stefan Jablonski	University of Bayreuth, Germany
Lili Jiang	Umea University, Sweden
Ahto Kalja	Tallinn University of Technology, Estonia
Mehmed Kantardzic	University of Louisville, USA
Dimitris Karagiannis	University of Vienna, Austria
Randi Karlsen	University of Tromsø, Norway
Zoubida Kedad	University of Versailles, France
Marite Kirikova	Riga Technical University, Latvia
Attila Kiss	Eötvös Loránd University, Hungary
Mikhail Kogalovsky	Market Economy Institute of the Russian Academy of Sciences, Russia
Margita Kon-Popovska	SS. Cyril and Methodius University Skopje, Macedonia
Michal Kopecký	Charles University, Prague, Czech Republic
Michal Kratky	VSB-Technical University of Ostrava, Czech Republic
Ulf Leser	Humboldt-Universität zu Berlin, Germany
Sebastian Link	The University of Auckland, New Zealand
Audrone Lupeikiene	Vilnius University Institute of Mathematics and Informatics, Lithuania
Gábor Magyar	Budapest University of Technology and Economics, Hungary
Christian Mancas	Ovidius University, Romania
Federica Mandreoli	University of Modena, Italy
Yannis Manolopoulos	Aristotle University of Thessaloniki, Greece
Manuk Manukyan	Yerevan State University, Armenia
Karol Matiaszko	University of Žilina, Slovakia
Brahim Medjahed	University of Michigan, Dearborn, USA
Bálint Molnár	Eötvös University of Budapest, Hungary
Tadeusz Morzy	Poznan University of Technology, Poland
Pavol Navrat	Slovak University of Technology, Slovakia
Martin Nečáský	Charles University, Prague, Czech Republic
Kjetil Norvøag	Norwegian University of Science and Technology, Norway
Boris Novikov	St. Petersburg University, Russia
Andreas Oberweis	Karlsruhe Institute of Technology, Germany
Andreas L Opdahl	University of Bergen, Norway
George Papadopoulos	University of Cyprus, Cyprus
Odyseas Papapetrou	DIAS, EPFL, Switzerland
András Pataricza	Budapest University of Technology and Economics, Hungary
Tomas Pitner	Masaryk University, Faculty of Informatics, Czech Republic
Jan Platos	VSB-Technical University of Ostrava, Czech Republic
Jaroslav Pokorný	Charles University in Prague, Czech Republic
Giuseppe Polese	University of Salerno, Italy
Boris Rachev	Technical University of Varna, Bulgaria

Miloš Radovanović	University of Novi Sad, Serbia
Heri Ramampiaro	Norwegian University of Science and Technology, Norway
Karel Richta	Czech Technical University, Prague, Czech Republic
Stefano Rizzi	University of Bologna, Italy
Peter Ruppel	Technische Universität Berlin, Germany
Gunter Saake	University of Magdeburg, Germany
Petr Saloun	VSB-TU Ostrava, Czech Republic
Shiori Sasaki	Keio University, Japan
Kai-Uwe Sattler	TU Ilmenau, Germany
Milos Savic	University of Novi Sad, Serbia
Ingo Schmitt	Technical University of Cottbus, Germany
Timos Sellis	Swinburne University of Technology, Australia
Maxim Shishaev	IIMM, Kola Science Center RAS, Russia
Bela Stantic	Griffith University, Australia
Kostas Stefanidis	University of Tampere, Finland
Claudia Steinberger	Universität Klagenfurt, Austria
Sergey Stupnikov	Russian Academy of Sciences, Russia
James Terwilliger	Microsoft, USA
Bernhard Thalheim	Christian Albrechts University, Kiel, Germany
Raquel Trillo-Lado	Universidad de Zaragoza, Spain
Olegas Vasilecas	Vilnius Gediminas Technical University, Lithuania
Goran Velinov	UKIM, Skopje, FYR Macedonia
Peter Vojtas	Charles University Prague, Czech Republic
Isabelle Wattiau	ESSEC and CNAM, France
Robert Wrembel	Poznan University of Technology, Poland
Weihai Yu	The Arctic University of Norway, Norway
Jaroslav Zendulka	Brno University of Technology, Czech Republic

Additional Reviewers

Irina Astrova, Estonia	Christos Mettouris, Cyprus
Dominik Bork, Austria	Patrick Schäfer, Germany
Antonio Corral, Spain	Jiří Šebek, Czech Republic
Senén González, Argentina	Jozef Tvarozek, Slovakia
Selma Khouri, Algeria	Theodoros Tzouramanis, Greece
Vimal Kunnummel, Austria	Goran Velinov, FYR Macedonia
Jevgeni Marenkov, Estonia	Alexandros Yeratziotis, Cyprus

Steering Committee

Chair

Leonid Kalinichenko Russian Academy of Science, Russia

Co-chair

Yannis Manolopoulos Aristotle University of Thessaloniki, Greece

Members

Paolo Atzeni	Università Roma Tre, Italy
Ladjet Bellatreche	LIAS/ENSMA, France
Andras Benczur	Eötvös Loránd University Budapest, Hungary
Maria Bielikova	Slovak University of Technology in Bratislava, Slovakia
Albertas Caplinskas	Vilnius University, Lithuania
Barbara Catania	DIBRIS University of Genova, Italy
Johann Eder	Alpen Adria Universität Klagenfurt, Austria
Janis Grundspenkis	Riga Technical University, Latvia
Hele-Mai Haav	Tallinn University of Technology, Estonia
Theo Haerder	TU Kaiserslautern, Germany
Mirjana Ivanovic	University of Novi Sad, Serbia
Hannu Jaakkola	Tampere University of Technology, Finland
Leonid Kalinichenko	Institute of Informatics Problems of the Russian Academy of Science, Russia
Marite Kirikova	Riga Technical University, Latvia
Mikhail Kogalovsky	Market Economy Institute of the Russian Academy of Sciences, Russia
Margita Kon-Popovska	SS. Cyril and Methodius University Skopje, Macedonia
Yannis Manopoulos	Aristotle University of Thessaloniki, Greece
Rainer Manthey	University of Bonn, Germany
Manuk Manukyan	Yerevan State University, Armenia
Joris Mihaeli	Israel
Tadeusz Morzy	Poznan University of Technology, Poland
Pavol Navrat	Slovak University of Technology, Slovakia
Boris Novikov	St. Petersburg University, Russia
Mykola Nikitchenko	Kyiv National Taras Shevchenko University, Ukraine
Jaroslav Pokorny	Charles University in Prague, Czech Republic
Boris Rachev	Technical University of Varna, Bulgaria
Bernhard Thalheim	Christian Albrechts University, Kiel, Germany
Gottfried Vossen	University of Münster, Germany
Tatjana Welzer	University of Maribor, Slovenia
Viacheslav Wolfengangen	Russia
Robert Wrembel	Poznan University of Technology, Poland
Ester Zumpano	University of Calabria, Italy

General Chair

András Benczúr Eötvös Loránd University, Budapest, Hungary

Program Chairs

Tomáš Horváth Eötvös Loránd University, Budapest, Hungary
Bernhard Thalheim University of Kiel, Germany

Proceedings Chair

Bálint Molnár Eötvös Loránd University, Budapest, Hungary

Workshops Chairs

Silvia Chiusiano Politecnico di Torino, Italy
Csaba Sildó SZTAKI (Institute for Computer Science and Control,
Hungarian Academy of Sciences), Budapest, Hungary
Tania Cerquitelli Politecnico di Torino, Italy

Doctoral Consortium Chairs


Michal Kopman Slovak University of Technology in Bratislava, Slovakia
Peter Z. Revesz University of Nebraska-Lincoln, USA
Sándor Laki Eötvös Loránd University, Budapest, Hungary

Organizing Committee

András Benczúr Eötvös Loránd University, Budapest, Hungary
Tomáš Horváth Eötvös Loránd University, Budapest, Hungary
Bálint Molnár Eötvös Loránd University, Budapest, Hungary
Anikó Csizmazia Eötvös Loránd University, Budapest, Hungary
Renáta Fóris Eötvös Loránd University, Budapest, Hungary
Ágnes Kerek Eötvös Loránd University, Budapest, Hungary
Gusztáv Hencsey Hungarian Academy of Sciences, Institute for Computer
Science and Control, Budapest, Hungary
Klára Biszkupné-Nánási Altagra Business Services and Travel Agency Ltd.,
Gödöllő, Hungary
Miklós Biszkup Altagra Business Services and Travel Agency Ltd.,
Gödöllő, Hungary
Judit Juhász Altagra Business Services and Travel Agency Ltd.,
Gödöllő, Hungary

Abstract of Invited Talks

Spatio-Temporal Data Mining of Major European River and Mountain Names Reveals Their Near Eastern and African Origins


Peter Z. Revesz 

University of Nebraska-Lincoln, Lincoln NE 68588, USA
revesz@cse.unl.edu

Abstract. This paper presents a spatio-temporal data mining regarding the origin of the names of the 218 longest European rivers. The study shows that 35.2 percent of these river names originate in the Near East and Southern Caucasus. The study also investigates the origin of European mountain names. It is shown that at least 26 mountain names originate from Africa.

Keywords: Data mining · Etymology · Mountain · River · Spatio-temporal

Database-Centric Scientific Computing (in Memoriam Jim Gray)

Alexander S. Szalay 

Department of Physics and Astronomy, Department of Computer Science,
The Johns Hopkins University, MD 21210, Baltimore, USA
szalay@jhu.edu

Abstract. Working with Jim Gray, we set out more than 20 years ago to design and build the archive for the Sloan Digital Sky Survey (SDSS), the SkyServer. The SDSS project collected a huge data set over a large fraction of the Northern Sky and turned it into an open resource for the world's astronomy community. Over the years the project has changed astronomy. Now the project is faced with the problem of how to ensure that the data will be preserved and kept alive for active use for another 15 to 20 years. At the time there were very few examples to learn from and we had to invent much of the system ourselves. The paper discusses the lessons learned, future directions and recalls some memorable moments of our collaboration.

Contents

Invited Papers

- Database-Centric Scientific Computing (In Memoriam Jim Gray) 3
Alexander S. Szalay
- Spatio-Temporal Data Mining of Major European River and Mountain
Names Reveals Their Near Eastern and African Origins. 20
Peter Z. Revesz

Information Extraction and Integration

- Query Rewriting for Heterogeneous Data Lakes 35
Rihan Hai, Christoph Quix, and Chen Zhou
- RawVis: Visual Exploration over Raw Data 50
*Nikos Bikakis, Stavros Maroulis, George Papastefanatos,
and Panos Vassiliadis*

Data Mining and Knowledge Discovery

- Extended Margin and Soft Balanced Strategies in Active Learning 69
Dávid Papp and Gábor Szűcs
- Location-Awareness in Time Series Compression 82
Xu Teng, Andreas Züfle, Goce Trajcevski, and Diego Klabjan

Indexing, Query Processing and Optimization

- Efficient SPARQL Evaluation on Stratified RDF Data with Meta-data. 99
Flavio Ferrarotti, Senén González, and Klaus-Dieter Schewe
- SIMD Vectorized Hashing for Grouped Aggregation 113
*Bala Gurumurthy, David Broneske, Marcus Pinnecke,
Gabriel Campero, and Gunter Saake*
- Selecting Sketches for Similarity Search. 127
Vladimir Mic, David Novak, Lucia Vadicamo, and Pavel Zezula
- On the Support of the Similarity-Aware Division Operator
in a Commercial RDBMS 142
Guilherme Q. Vasconcelos, Daniel S. Kaster, and Robson L. F. Cordeiro

Data Quality and Data Cleansing

Data Quality in a Big Data Context. 159
Franco Arolfo and Alejandro Vaisman

Integrating Approximate String Matching with Phonetic String Similarity. 173
Junior Ferri, Hegler Tissot, and Marcos Didonet Del Fabro

Distributed Data Platforms, Including Cloud Data Systems, Key-Value Stores, and Big Data Systems

Cost-Based Sharing and Recycling of (Intermediate) Results in Dataflow Programs 185
Stefan Hagedorn and Kai-Uwe Sattler

ATUN-HL: Auto Tuning of Hybrid Layouts Using Workload and Data Characteristics. 200
Rana Faisal Munir, Alberto Abelló, Oscar Romero, Maik Thiele, and Wolfgang Lehner

Set Similarity Joins with Complex Expressions on Distributed Platforms 216
Diego Junior do Carmo Oliveira, Felipe Ferreira Borges, Leonardo Andrade Ribeiro, and Alfredo Cuzzocrea

Streaming Data Analysis

Deterministic Model for Distributed Speculative Stream Processing 233
Igor E. Kuralenok, Artem Trofimov, Nikita Marshalkin, and Boris Novikov

Large-Scale Real-Time News Recommendation Based on Semantic Data Analysis and Users’ Implicit and Explicit Behaviors 247
Hemza Fichel, Mohamed Ramzi Haddad, and Hajer Baazaoui Zghal

Web, XML and Semi-structured Databases

MatBase Constraint Sets Coherence and Minimality Enforcement Algorithms 263
Christian Mancas

Integration of Unsound Data in P2P Systems 278
Luciano Caroprese and Ester Zumpano


Author Index 291

Invited Papers



Database-Centric Scientific Computing

(In Memoriam Jim Gray)

Alexander S. Szalay^(✉) 

Department of Physics and Astronomy, Department of Computer Science,
The Johns Hopkins University, Baltimore, MD 21210, USA
szalay@jhu.edu

Abstract. Working with Jim Gray, we set out more than 20 years ago to design and build the archive for the Sloan Digital Sky Survey (SDSS), the SkyServer. The SDSS project collected a huge data set over a large fraction of the Northern Sky and turned it into an open resource for the world’s astronomy community. Over the years the project has changed astronomy. Now the project is faced with the problem of how to ensure that the data will be preserved and kept alive for active use for another 15 to 20 years. At the time there were very few examples to learn from and we had to invent much of the system ourselves. The paper discusses the lessons learned, future directions and recalls some memorable moments of our collaboration.

1 Introduction

The unprecedented amounts of observational, experimental and simulation data are transforming the nature of scientific research. As more and more data sets are becoming public, we need to find the right ways not just to make the data public, but accessible and usable. Our techniques have not kept up with this evolution. Simple things like moving large amounts of data to the computing are becoming difficult; as a result, scientists are learning how to “move the analysis to the data”.

The data from Sloan Digital Sky Survey (SDSS) has been one of the best examples of a large-scale, open scientific data set. It has been in the public domain for almost twenty years [1, 2]. The project and its archive have changed astronomy forever. It has shown that a whole community is willing to change its traditional approach and use a virtual instrument, if the data is of high quality, and if it is presented in an intuitive fashion. The continuous evolution and curation of the system over the years has been an intense effort, and has given us a unique perspective of the challenges involved in the broader problem of operating open archival systems over a decade.

The SDSS is special – it has been one of the first major open e-science archive. Tracking its usage as well as the changes that occurred helps the whole science community to understand the long-term management of such data sets, and see what common lessons can be derived for other disciplines facing similar challenges. These archives not only serve flat files and simple digital objects, but they also present complex services, based on multi-terabyte databases. The toolkits (and even the underlying operating systems) change, the service standards evolve, and even though

some services may have been cutting edge ten years ago, today they may become dated. New protocols emerge, and to support the increasingly sophisticated client-side environments, the services need active updates at regular intervals.

Scientists in many disciplines would like to compare the results of their experiments to data emerging from numerical simulations based on first principles. This requires not only that we are able to run sophisticated simulations and models, but that the results of these simulations are available publicly, through an easy-to-use portal. We have to turn the simulations into open numerical laboratories where anyone can perform their own experiments. Integrating and comparing experiments to simulations is another non-trivial data management challenge, in the same spirit as in the SDSS archive. Not every data set from these simulations has the same lifecycle. Some results are just transient and need to be stored for a short while to analyze, while others will become community references, with a useful lifetime of a decade or more.

With the largest supercomputers approaching Exascale, with memories in peta-bytes, it will soon become impossible to simply use memory dumps as the sharable scientific output. We need a very different strategy, of finding the most interesting events while the simulations are running on the supercomputers, and triggering a “storage event”, where only a very small number of discrete, localized regions of the simulations are written to disk. The challenge is to find a strategy to make sure that these regions represent the highest scientific value for posterior analyses. Their access will be no longer sequential, but through complex multi-dimensional indices.

A common theme in these projects has been the systematic use of databases as the underlying access medium. This paper will discuss the journey started with our collaboration with Jim Gray, and how we followed this path since then. The last two decades have seen a remarkable evolution in distributed computing, when we went from a handful of systems running in parallel to millions of machines in the cloud, from tens of Gigabytes to Petabytes and Exabytes. New computational paradigms came and went. Many new flavors of databases emerged. It is clear today that databases (SQL or noSQL) have many unique features to offer, for both commercial and scientific applications, and they will be a big part of the future.

New technologies, like streaming algorithms, machine learning, Deep Learning, neuromorphic computing, accelerators like GPUs and FPGAs will all soon be part of the database universe. The sharp boundaries between databases and parallel computing are already dissolving, and even filesystems are being replaced by clever uses of massive databases.

In this paper I would like to remember Jim Gray, a wonderful friend and collaborator, who has seen all these profound changes coming. He gave the opening keynote on ADBIS 2006, also held in Budapest. I would like to use this opportunity to talk about the story of our collaboration, and discuss the fundamental changes Jim’s contributions brought to science.

2 Jim Gray and the SDSS SkyServer

Alex Szalay and Ani Thakar, in collaboration with Jim Gray (Microsoft) have spent much of the last two decades working on the archive for the Sloan Digital Sky Survey. The archive was originally built on top of an object-oriented database, but after a few years it became clear that the users will want to have a very flexible query environment with a lot of ad-hoc queries. The promised flexible Object Query Language (OQL) was very slow to emerge and turned out to be very fragile. As a result, we have started to develop our own (very limited) query language (Fig. 1).



Fig. 1. Jim Gray with members of our team. From left to right: Alex Szalay, Tamas Budavari, Jim Gray and Ani Thakar.

It was around this time, when we met Jim Gray of Microsoft. Jim liked to say, that the “best collaborators are the desperate ones”, as they are ready to change the way they approach a problem. We were definitely desperate at this point. After a few meetings Jim advocated that we should consider using a relational database. He made the point that a few programmers in an academic environment cannot successfully compete with the thousands of developers at Microsoft, Oracle and IBM, and we should spend our efforts on creating the additional “business logic” related to astronomy, and use a commercial platform with a robust SQL query engine. This advice set us on the trajectory that we have followed ever since.

Another major principle behind our approach also came from Jim Gray. When we first met, he asked me to give the “20 queries” astronomers wanted to ask from the database. My first (kneejerk) reaction was that we would like to ask anything and everything, and this is what science is about. Jim just smiled, and asked me to give the first five that came to mind. I quickly wrote the down. He then asked me to give the next five. It took the better part of an hour to do so. By then I realized that the next five

will take much longer, and understood that not every question is the same. In an hour this methodology has taught me a lot of humility, and I learned that there is a priority among all the things that we can conceivably think of. We have since used this heuristic technique in a lot of different projects and settings. The results are always the same: after an initial astonishment the domain scientists very quickly “get the idea” of establishing clear priorities, and the database architect and domain scientists quickly find the common ground.

The project has revolutionized not just professional astronomy but also the public’s access to it. Although a substantial portion of the astronomy community is using the SDSS archive on a daily basis, the archive has also attracted a wide range of users from the public [3]: a recent scan of the logs show more than 4M distinct IP addresses accessing the site. The total number of professional astronomers worldwide is only about 15K. Furthermore, the collaborative CasJobs interface has more than 9,000 registered users - almost half of the professional astronomy community.

SDSS (2000–2005) and its successors SDSS-II (2005–2008) and SDSS-III (2008–2014) have an extraordinary legacy of mapping structure across a vast range of scales, from asteroids in our own Solar System to quasars over 10 billion light-years away. These surveys have produced data to support the 7,000 papers with more than 450,000 citations. The SDSS has several times been named the highest impact project, facility or mission in the field of astronomy, as judged by number of citations of associated refereed journal articles [4, 5].

The SDSS was the source of the most highly cited astronomy article in the years 2000, 2002, 2005, and 2008 [6]. Within the Collaboration there have been over 120 SDSS-based PhD theses, and outside the Collaboration there have been many more. Its publicly available, user-friendly tools have fueled a large number of undergraduate and

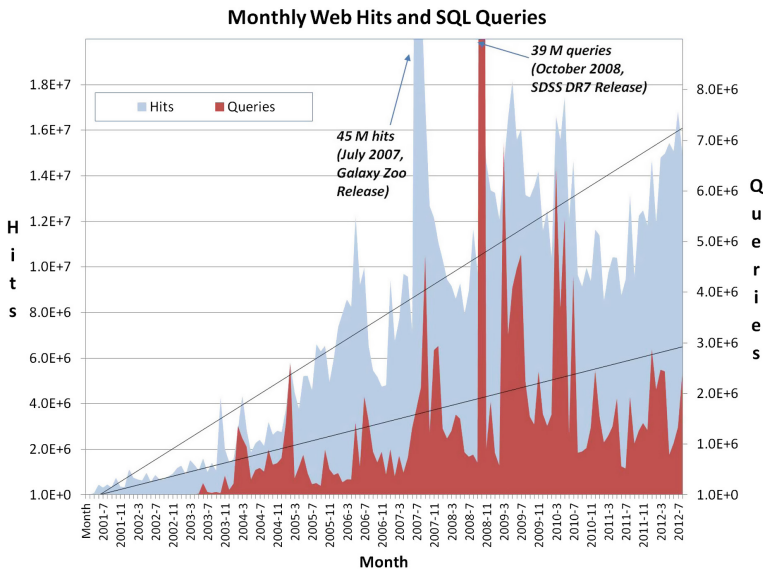


Fig. 2. Monthly traffic of the SDSS SkyServer

even high-school projects and research papers (including Siemens Competition winners). Galaxy Zoo, based on SDSS, is a hugely successful “citizen science” experiment that has involved over 100,000 members of the public in astronomical discovery. A key component of its success has been the cutting-edge data distribution system provided through SkyServer and CasJobs. The most recent version of the database has a 15 TB queryable public core, with about 150 TB additional raw and calibrated files [7]. The traffic on the website is still growing, recently we had to switch the log database to 64-bit counters, as some of the counter values have exceeded the 32-bit range.

3 Emerging Innovations in the SkyServer

3.1 The SDSS Web Services

The web interface is built on a very small number of atomic services. One simply executes a single SQL query. This service was very carefully written, it parses the body of the query looking for all sorts of ‘bad’ patterns, like injection queries, DROP TABLE, or ALTER TABLE like DDL statements. This function also heavily interacts with the log, it writes the SQL string into the database when the query is started, then logs the lapse and CPU times, and the number of rows delivered, or the error code if any.

The other major core service is the ImageCutout [8]. This is computing the displayed views of the sky dynamically, using a multiresolution set of the more than a million SDSS images. This code has been written in C#, and uses the Microsoft .NET GDI library for the warping of images. Given the enormous dynamic range in the brightness of astronomical objects, we used a special color-preserving as in transformation of the pixel values [9].

3.2 The CasJobs/MyDB Collaborative Environment

As traffic on the SDSS archive grew, many users were running repeated queries extracting a few million rows of data. The DB server delivered such data sets in 10 s, but it took several minutes to transmit the data through the slow wide-area networks. We realized that if users had their own databases at the server, then the query outputs could go through a high-speed connection, directly into their local databases, improving system throughput by a factor of 10. During the same time, typical query execution times and result sets kept growing, thus synchronous, browser-based access was no longer enough, especially for the rapidly growing segment of “power users”. There had to be an asynchronous (batch) mode that enabled queries to be queued for execution and results to be retrieved later at will.

The CasJobs/MyDB batch query workbench environment was born as a result of combining these “take the analysis to the data” and asynchronous query execution concepts. CasJobs builds a flexible shell on top of the large SDSS database (Fig. 3).

Users are able to conduct sophisticated database operations within their own space: they can create new tables, perform joins with the main DB, write their own functions, upload their own tables, and extract existing value-added data sets that they can take to their home environment, to be used with the familiar tools they have been using since their graduate years. The system became an overnight success. Users needed to register, and each received an initial 0.5 GB database. Their data and query history is always available to them through their user id, and they do not need to remember things like “how did I create this data set?”

For redundancy, we had three identical servers containing the active databases. By studying the usage patterns, we realized that the query length distribution was well represented by a power law. Hence, we split the traffic into multiple queues served by different servers, each handling the same aggregate workload [10]. Each query can be submitted to a “fast,” a “long,” or a MyDB queue, returning the result into a MyDB table. The user can then process the derived result further, run a multi-step workflow, or extract the data. Everything that a user does is logged. This set of user-controlled databases form a very flexible tier on top of the rigid schema of the archive. This resolves the long-standing tension between stability and rigidity for the integrity of the core data, and the flexibility to provide room for user creativity.

As users became familiar with the system, there were requests for data sharing. As a result, we added the ability to create groups and to make individual tables accessible to certain groups. This led to a natural self-organization, as groups working on a collaborative research project used this environment to explore and build their final, value-added data for eventual publication. GalaxyZoo [11], which classified over a million SDSS galaxies though a user community of 300,000, used CasJobs to make the final results world-visible, and CasJobs also became a de-facto platform for publishing data. We added the capability for users to upload their own datasets and import them into their MyDBs for correlation with the SDSS. As depicted in Fig. 2, CasJobs-based systems are currently being used as the primary means of serving the catalog data for the GALEX and Pan-STARRS projects (among others) in astronomy, in addition to serving the catalog data for multiple SDSS generations and their worldwide mirrors. CasJobs has also been (or is currently being) adapted for simulation data projects and non-astronomy datasets.

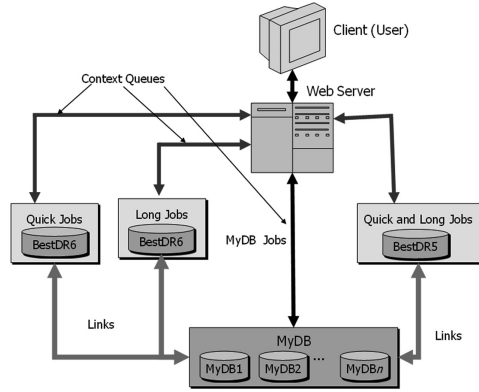


Fig. 3. The CasJobs/MyDB framework built for the Sloan Digital Sky Survey archive.

3.3 SQL Extensions

In our collaboration with Jim Gray, we have developed a Design Pattern to add domain specific extensions to SQL Server, using CLI integration. Our code for spatial indexing was used in the “shrink-wrap” production version of SQL Server 2005 [12–14]. This code also formed the basis of the subsequent Microsoft Spatial Index. The idea is to take a class library written in one of the .NET languages (C++, Java, C#), store a binary instance of the class as a binary datatype, and expose the object methods as user-defined functions (UDFs). SQL Server makes this very convenient, since unlike many other database platforms like MySQL, it allows for table-valued UDFs. One can then pass the binary object as a parameter to the function and execute the method, or access the property.

In the SkyServer we have 236 UDFs supporting detailed astronomy knowledge, like conversion of cosmological coordinates in a curved spacetime to angles and radial distances. Also, we have built an astronomy-specific spatial index, capable of representing spherical polygons with milliarcsec accuracy over the whole sky, and with a relational algebra over the regions, and fast indexing capabilities finding several million points per spherical region in a second.

This turned into another generic indexing pattern. Following the algorithms in the seminal book of Samet [15] we have built a Euclidian version of the spatial index in 2D and 3D, using various space filling curves as plugins sharing the same class [16]. Our turbulence database is using a Morton (or Z) curve, the cosmological simulations use a Peano-Hilbert curve. Then we map the resulting hash code onto a B-tree index in the relational database. The points in each bucket along the space filling curve at a given granularity then map onto a range query, performed extremely fast inside a relational database. The range of buckets to be searched is given by the intersection of the geometric search volume with the quad- or oct-tree of the space filling curve. This is another component that is quite generic. We need to add additional geometric primitives to the library.

For large numerical simulations much of the data is in multidimensional floating-point arrays. We have built such a User Defined Type for SQL Server, which is used for all of our simulation databases [17]. We will develop a generic module that repartitions the data in a large array into smaller blocks organized along a space-filling curve, adds the custom metadata header and writes these out in native binary format for optimal SQL Server load performance.

3.4 Schema and Metadata Framework

The schema for the database is contained in a set of DDL files. These files are quite complex, they not only contain the code to generate the database and the associated stored procedures and user defined functions, but in the comment fields of the scripts they contain rich metadata describing the schema elements, including physical units, enumerations, indexes, primary keys, short and long descriptions. A parser can extract this information at different granularities (object, column) and create a set of metadata tables that can be automatically loaded into the database [18]. This ensures that all the schema and related metadata is handled together in an automated fashion, similar to the

approach originally employed by Donald Knuth, when he created TeX. The database will then contain all the up-to-date metadata and these can be queried and displayed using simple functions and dynamic web services. This tool is quite robust and mature and has been in use for more than 14 years.

4 Extending to Other Disciplines

4.1 The SkyServer Genealogy

The template for the SDSS archive is now being used within astronomy by several projects and institutions beyond JHU (STScI, Fermilab, Caltech, Edinburgh, Hawaii, Portsmouth, and Budapest). The technologies and concepts used for the SDSS archive have since been used beyond astronomy. Using the same template, we have built databases for a growing number of other disciplines; see Fig. 2. for the genealogy of SDSS. Included are databases for turbulence [19, 20], radiation oncology [21], environmental sensing and carbon cycle monitoring [22] and most recently a prototype for high-throughput genomics. The databases built for cosmological simulations at the Max Planck Institutes in Garching [23] and Potsdam are revolutionizing how astronomers interact with the largest simulations. We would also like to acknowledge that the prototype of the SkyServer was partly derived from the Terraserver, written by Jim Gray and Tom Barclay (Fig. 4).

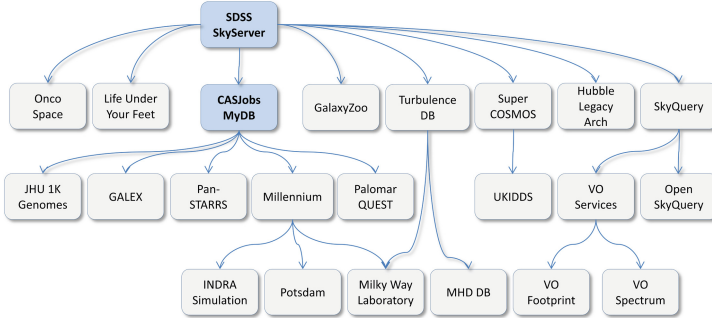


Fig. 4. The genealogy of the SDSS archive and the CasJobs/MyDB system.

4.2 Database-Centric Open Numerical Laboratories

The Emerging Simulation Data Challenge. World-wide there is an ongoing effort to build an exascale computer. Some of the largest particle simulations today already exceed a trillion particles. If we only store their positions and velocities, and a single particle identifier, we have to save 56 TB for each snapshot. At the same time, fewer and fewer codes will scale to run on millions of cores, and as a result, fewer and fewer people will use these machines. There will be an increasing gap between the wide science community and the top users. It will be increasingly important to be able to

create tangible benefits, usable science products that can be used by a much broader pool of users: otherwise community support will be soon endangered. Thus, we postulate that it is extremely important to identify a mechanism through which data products from the largest simulations in science can be publicly shared and used, potentially over extended periods. Databases play an important role in these applications.

Databases and Immersive Analysis Through Virtual Sensors. For a large-scale analyses we need a data access abstraction that is inherently scalable. For the user it should not matter whether the data in is a terabyte or a petabyte. The casual user should be able to perform very light-weight analyses, without downloading much software or data. Accessing data through the flat files violates this principle: the user cannot do anything until a very large file has been physically transferred.

On the other hand, one can create a so-called *immersive environment* in which the users can insert virtual sensors into the simulation. These sensors can then feed data back to the user. They can provide a one-time measurement, pinned to a physical (Eulerian) location or they can “go with the flow” as comoving Lagrangian particles. As a result of this access pattern, the only scaling is related to the number of sensors. The data access is extremely localized and a relational database is providing the data access layer, using distributed SQL queries, turning the spatial hash keys into delivering buckets of data, used for the interpolation of data values as part of a User Defined Function.

By placing the sensors in different geometric configurations, users can accommodate a wide variety of spatio-temporal access patterns. The sensors can feed back data on multiple channels, measuring different fields in the simulation. They can have a variety of operators, like the Hessian or Laplacian of a field, or various filters and clipping thresholds. This design also enables the users to run time backwards, impossible in a direct simulation involving dissipation. Imagine that the snapshots are saved frequently enough that one can interpolate particle velocities smoothly enough. Sensors can back-track their original trajectory and one can see where they came from, all the way back to the initial conditions. This simple interface can provide a very flexible, yet powerful way to do science with large data sets from anywhere in the world.

What is noteworthy about the availability of such a 4D dataset “at your fingertips” and the ability to make “casual” queries from anywhere, at any time, is that it is beginning to change how we think about the data. Researchers can come back to the same place in space and time and be sure to encounter the same values. They can follow and observe phenomena forward and backwards in time.

The Turbulence Database. The “Twister” metaphor mentioned above, which is based on our “immersive turbulence” paradigm, has been implemented in the Turbulence DB 8 years ago. The Turbulence DB is the first space-time database for turbulent flows, containing the output of large simulations, publicly available to the research community, [24, 25]. The 27 TB database contains the entire time-history of a 1024^3 mesh point pseudo-spectral Direct Numerical Simulation of forced Navier-Stokes equations representing isotropic turbulence. 1024 time-steps are stored, covering a full “large-eddy” turnover time of model evolution. We have built a prototype web service

[24–26] that serves requests over the web for velocities, pressure, various space derivatives of velocity and pressure, and interpolation functions. The data and its interface are used by the turbulence research community and have led to about 100 publications to date. To date we have delivered over 70 trillion (!) data points to the user community.

A dataset on Magneto-Hydrodynamic Turbulence (50 TB) was the second one added to the database. After this, we have added a channel flow database (120 TB), and a smaller database of a rotating fluid. Then, we have

augmented the original isotropic turbulence from 1024 snapshots to 5028 snapshots, increasing its size to well over 100 TB. We are in the process of converting and loading another MHD database with a volume of 256 TB. Figure 5 shows a few thousand particles on stochastic trajectories, all ending within a small distance to each other. The trajectories were computed by moving the particles backward in time, impossible to do in an in-situ computation, only enabled by interpolation over the database.

Evaluation of filtered or coarse-grained fields requires intermediate, server-side computations, such as FFTs or streamed partial sum evaluations of linear operators acting on the data. The turbulence data sets are simply too big to transfer wholesale to the user, and conversely, the “pre-determined” user defined functions that can be implemented within the database system close to the data are simply too inflexible to enable the next steps of data analysis we wish to perform on the data.

We are in the process of adding CasJobs/MyDB/MyScratch functionality to the turbulence databases, enabling users to run batch jobs, store and correlate these intermediate derived data sets with the main data. Furthermore, we will develop two specific scalable algorithms that capture the most important patterns to create custom intermediate data sets within the application server layer. We have made some initial steps already [28], but much more work is required to make these tools sufficiently robust and efficient. For repeated analyses we will also enable the users to store/cache such results in their MyDB for reuse.

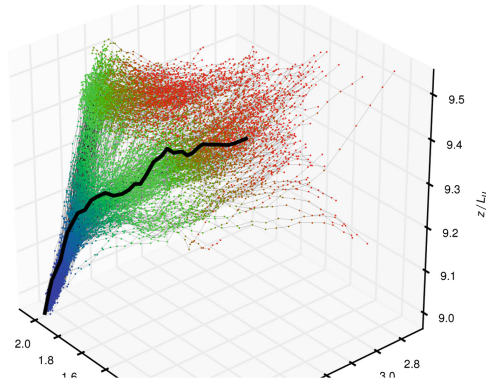


Fig. 5. Stochastic trajectories of particles with a common final position from a 50 TB MHD simulation. The particle tracking has been performed backwards in time, impossible to do in an in-situ CFD simulation due to the dissipative nature of the problem [27].

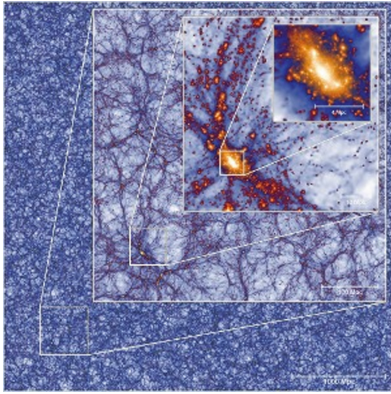


Fig. 6. Zoom in on a slice through the Millennium-XXL simulation [29].

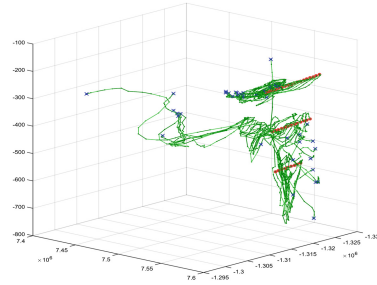


Fig. 7. Three-dimensional particle trajectories in an ocean circulation model in the North Atlantic region. The particles are inserted posterior, running in a MATLAB script, through a Jupyter interface.

Cosmological N-body Simulations. A similar transformation is happening in cosmology. The SDSS SkyServer framework was reused by Gerard Lemson at the Max Planck Institute in Garching, Germany for the Millennium simulation database. The database has been in use for over 8 years, and has hundreds of regular users, and has been used in nearly seven hundred publications. The database contains value added data from a simulation originally only containing 10B dark matter particles. These particles form hierarchical clusters, so called sub-halos and halos, which in turn become the sites where galaxies might form. A semi-analytical recipe was used to create mock galaxies in the simulations, and their hierarchical mergers were tracked in a database structure. The merger history was used to assign a plausible star formation rate to each galaxy which in turn can be used to derive observable physical properties. The database contains several such semi-analytic scenarios and has been expanded with data from three other simulations, one of which, the Millennium XXL [29] is containing 300 billion particles (Fig. 6).

It is clear, that there is a similar momentum building in the cosmology community as in turbulence, and by being able to move quickly, we can maintain a world-wide leading role in this emerging field. In order to maximize the access performance to the data, the halos and particles are stored along a space-filling Peano-Hilbert curve in the database, which creates a linear ordering maximally coherent in 3D. As hard disks are inherently sequential devices, this maximizes the database performance (Fig. 7).

Kilometer-Scale Modeling of Virtual Ocean Basins. Tom Haine and his postdocs and students are developing basin scale models of the ocean with kilometer scale resolutions. Such models, with appropriate numerical schemes, can help test fundamental ideas about how the vast range of turbulent scales in the ocean interact to maintain climate and how the dominant biomass processes in the ocean respond to and are affected by the rich ocean eddy field. Many key questions remain unanswered in these areas and their answers have important implications for the role the ocean can

play in stabilizing or destabilizing the Earth’s thermal, hydrologic, chemical and biological balances on time scales from days to decades and beyond.

5 The Next Steps: From SkyServer to SciServer

5.1 The SciServer Architecture

Until a few years ago the SDSS framework was enhanced and expanded to include other types of data in a rather ad-hoc fashion. The high-level system architecture is shown on Fig. 8. The system diagram shows (at a very high level) how the different components interact. The diagram reflects also how the CasJobs/MyDB component is integrated with the SciDrive object store and how the scratch spaces (both flat files and databases) are linked to the system, and how the virtual machines are used to support interactive Jupyter-based scripting.

Over the first 12 years of the SDSS archive we have incrementally evolved the system, avoiding major architectural changes. The SDSS data with all the additional science projects have been created at a cost well over \$200M. They are widely used by a large community, generating new papers and supporting original research every day.

Now, as we need to look ahead into the future, the services are showing signs of a Service Lifecycle, while the data are still very much alive, they will still be used in 15 years from now. In order to prepare for the future, we need to consolidate and reengineer the services framework, with the main goal of making the data services much more sustainable, and inexpensive to operate. In order to do this, we have endeavored on converting the SkyServer to the SciServer, a more generic, modular framework of a set of building blocks that can be connected in several ways. In this section we will show the ideas behind this consolidation process, and how we add new services to fulfill emerging new needs from our users.

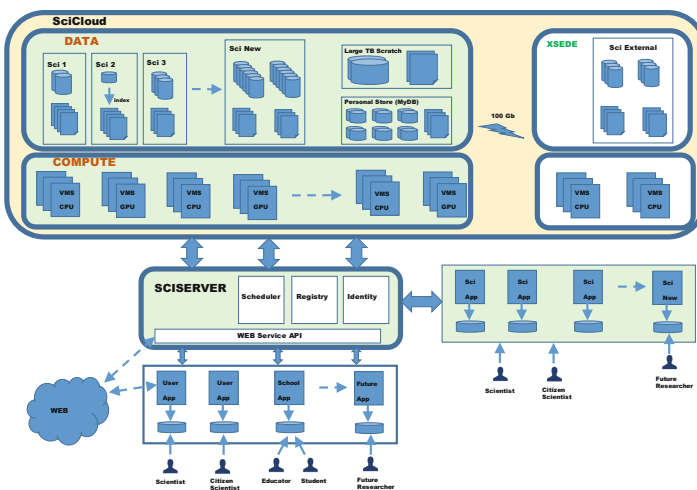


Fig. 8. The architectural diagram of the SciServer

Major challenges exist, from basic cost-effective engineering of sufficiently reliable, high-performance storage subsystems to productive, scalable analysis systems, collectively capable of tens to hundreds of GBps per second throughput and 1000 to 10,000-way parallelism. In this realm we have been experimenting with distributed database architectures that include enhanced query languages and data models that understand spatio-temporal data.

5.2 New Building Blocks

The Millennium and the Turbulence databases have used space filling curves as indexes inside a relational database. This gave an excellent indexed access performance, maximally parallel due to the query optimizer, and enabling the users to access the data through a declarative interface: they did not have to specify how to set up the iterators through the data over a distributed set of storage servers, this was hidden by the query engine. This database-centric view became very powerful, but it did not come for free. Loading buckets of data in binary BLOBs in the turbulence database was simple for small datasets, up to about a few tens of TB is data volume. Beyond that scale the process did not scale well at all, ingesting a 250 TB simulation took several months.

Over the last few years, Gerard Lemson has developed a new data organization for cosmological N-body simulations, which keeps the benefits of the database indexing but avoids the cumbersome database loading. This idea, **FileDB** is that we keep the data in (where possible) the original large binary files in the file system, but internally the data is split into buckets ordered along a space-filling curve. The files can be read in their entirety through the operating system for global analytics, but for localized access we can keep an index in a (small) database, which tells us the file and the offset of each data bucket/item. We only need to load the index into the database.

One table contains pointers to the file locations and has metadata concerning the snapshot and index range. A second table stores for each file the precise start and end point of each bucket. Querying for a certain bucket consists of finding the file and offset in that file. A seek operation moves one efficiently, without requiring any I/O, to the start point. A sequential read to the end of the bucket will load all the particles into memory.

This organization on a uniform, hierarchical, recursively indexed grid is the basis for an efficient implementation of queries for particles in spatial regions such as boxes, or spheres. It is quite straightforward to calculate all the cells overlapping such a region. The same pattern has now been applied to the latest turbulence data, which are no longer stored inside the database. Their access APIs still use the same database procedures and functions as before, but instead of querying the database tables these functions use the known spatial organization of the files to perform efficient seek-and-scan to retrieve the individual data points.

In that case the access is simplified as the data is uniform and exact locations of requested points can be calculated in closed form. This is in contrast to the particle data in the cosmological simulations that requires indexing of each separate snapshot. The oceanography case falls somewhere in between. The grids are known and constant, but

the irregular boundary conditions prevent one from closed form solutions, and a special index must be built inside the database as well.

We have enabled the CasJobs system to have many other *contexts*, not just the SDSS data versions (right now we have all the previous data releases from DR1 through DR9), but also other astronomical collections. We have also brought the simulations into the federation. Uploaded and derived data (and the related metadata) automatically shows up in the user’s MyDB. For large scale intermediate data sets the few GB of user space provided by default is not quite enough. For example, a custom cross-match of large astronomical catalogs, like SDSS and GALEX might require several 100 GB if not TBs of disk space. This cannot be done today. We aim resolved this problem by a new *MyScratch* layer between the static contexts and the MyDBs, with 200 TB of storage. In order to make it easier for users to upload their tables to the CasJobs environment, we will add a simple DropBox-like drag-and-drop interface to the system.

Our users, both in SDSS and in the Numerical Laboratories have become quite artful in using database queries. They use SQL tools not as a hammer, but rather as a violin, and they generate “nice music”. But, with the emergence of Python, lot of sophisticated machine learning algorithms, libraries and packages have become available, and the users are now keen to use these with same ease of interactivity as SQL. A typical use case would start with a SQL query returning tens of thousands of objects with a particular spectral property. However, the user would then like to go back to the raw data (spectra in this case) and run her own tools and algorithm written in Python.

In order to facilitate this, we built a new component: the *SciServer Compute*, a set of servers providing about 100 virtual machines, always available, that can be used to start Jupyter/iPython notebooks, within Docker containers. These are preconfigured with the database interface tools, that users can run their SQL queries out of Python. Furthermore, all the raw data files of SDSS (about 150 TB) are wrapped into a data container, so access is trivial. Other large projects have similar data containers. The Jupyter environment also enables Matlab and R, more relevant for our engineering and Biostats/genomics users. Several of our interactive Numerical Laboratories (Turbulence, Ocean Circulation, N-body) are now using both Python and Matlab bindings.

6 Futures of Database-Centric Computing

What has been emerging in all these use cases is that databases have been used as an extremely efficient API for localized, parallel I/O streams. Over the last few years there has been more evolution in terms of storage hardware than for several decades before. Today’s solid-state disks (SSDs) are commonplace, although expensive, but soon will be much bigger and much cheaper. One can already buy 16 TB SSDs. NVRAMs will add another new tier into the storage hierarchy.

As the underlying hardware gets increasingly more complex, the declarative programming interface of databases isolates the users from writing explicit iterators, and the query optimizers provide an efficient parallelism over the distributed hardware.

Big Data is always noisy, full with systematic errors – thus perfect answers are meaningless. In a massively parallel system, executing over possibly hundreds of thousands of data partitions, there will always be a few which will lag behind in execution. With databases starting to go beyond several petabytes, and with extreme parallelism, providing a statistically well defined, approximate result using sublinear, randomized algorithms may be a way to trade accuracy versus execution time in a controlled fashion. Sketches, approximate aggregations, may provide excellent trade-offs for heavy-hitter problems. Extreme value statistics may offer additional insights into Top-k queries.

POSIX file systems are providing a byte stream, completely hiding data locality. For efficient data queries, locality is important, and increasingly we see intelligent object stores offering more and more database-like features. Spanner and BigTable have changed how Google deals with their distributed data objects. It is safe to say that in some form we see databases gradually replacing file systems.

New computational hardware is emerging as well. Neuromorphic chips enable extremely fast execution of neural net training and prediction. Preliminary results show that Deep Learning can predict the locality of a data item based on its key value better than a B-tree [30, 31]. This is easy to understand when we consider an index to be simply a mapping of the cumulative distribution of the key values onto an address space. Such a curve fitting problem maps onto Deep Learning extremely well. Soon we will see Deep Learning also replace the decision trees of query optimizers [32]. GPU based sorting is many orders of magnitude faster than CPUs. Soon these devices will also be running database kernels, not just user-space application codes. There is more innovation, basic research and new platforms for databases emerging, than any time in the past, underlining the importance of data locality.

7 Summary

Relational databases have been around for a long time. Over the last few years we have seen a dramatic change in storage hardware, in distributed computing and these have resulted in profound changes, many new features in database architectures. Recently we have seen how databases are moving to the core of scalable data-intensive computing.

Many sciences now deploy their data into large, open scientific databases, hosting “smart” data, through computational services. This approach is embraced by the scientists, and enables them to manage and analyze data on scales previously unheard of. Data science is becoming as important as mathematics for every scientist. Jim Gray’s Fourth Paradigm is in full bloom, and he has foreseen many of the technological and scientific advances we see today.

These large data sets, analyzed by a much broader range of scientists than ever before, using all the tools of the computer age are creating a new way to do science, one that we are just starting to grapple with. We cannot predict where it will exactly lead, but it is already clear that these technologies will bring about further dramatic changes in the way we do science and make new discoveries.

References

1. SDSS SkyServer. <http://skyserver.sdss.org/>
2. Szalay, A.S., Kunszt, P., Thakar, A., Gray, J., Slutz, D., Brunner, R.: Designing and mining multi-terabyte astronomy archives: the sloan digital sky survey. In: Proceedings of the SIGMOD 2000 Conference, pp. 451–462 (2000)
3. Singh, V., et al.: SkyServer traffic report – the first five years. Microsoft Technical report, MSR-TR-2006-190 (2006)
4. Michael Banks: Impact of Sky Surveys. *Physics World*, p. 10, March 2009
5. Madrid, J.P., Macchetto, F.D.: High-impact astronomical observatories. *Bull. Am. Astron. Soc.* **41**, 913–914 (2009)
6. Frogel, J.A.: Astronomy’s greatest hits: the 100 most cited papers in each year of the first decade of the 21st century (2000–2009). *Publ. Astron. Soc. Pac.* **122**(896), 1214–1235 (2010)
7. Aihara, H., Allende Prieto, C., An, D., et al.: The eighth data release of the Sloan Digital Sky Survey: first data from SDSS-III. *Astrophys. J. Suppl.* **193**, 29 (2011)
8. Thakar, A.R., Szalay, A., Fekete, G., Gray, J.: The catalog archive server database management system. *Comput. Sci. Eng.* **10**, 30 (2008)
9. Lupton, R.H., Gunn, J.E., Szalay, A.S.: A modified magnitude system that produces well-behaved magnitudes, colors, and errors even for low signal-to-noise ratio measurements. *Astron. J.* **118**, 1406L (1999)
10. O’Mullane, W., Gray, J., Li, N., Budavari, T., Nieto-Santisteban, M., Szalay, A.S.: Batch query system with interactive local storage for SDSS and the VO. In: Ochsenbein, F., Allen, M., Egret, D. (eds.) Proceedings of the ADASS XIII, ASP Conference Series, vol. 314, p. 372 (2004)
11. Lintott, C.J., et al.: Galaxy Zoo: morphologies derived from visual inspection of galaxies from the Sloan Digital Sky Survey. *Mon. Not. R. Astron. Soc.* **389**, 1179–1189 (2008). <https://doi.org/10.1111/j.1365-2966.2008.13689.x>
12. Gray, J., Szalay, A.S., Fekete, G.: Using table valued functions in SQL server 2005 to implement a spatial data library. MSR-TR-2005-122 (2005)
13. Fekete, G., Szalay, A.S., Gray, J.: Using table valued functions in SQL server 2005. MSDN Development Forum (2006)
14. Budavári, T., Szalay, A.S., Fekete, G.: Searchable sky coverage of astronomical observations: footprints and exposures. *Publ. Astron. Soc. Pac.* **122**, 1375–1388 (2010)
15. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan-Kaufman, Burlington (2006)
16. Lemson, G., Budavari, T., Szalay, A.S.: Implementing a general spatial indexing library for relational databases of large numerical simulations. In: Proceedings of the SSDBMS Conference, Portland OR (2011)
17. Dobos, L., et al.: Array requirements for scientific applications and an implementation for microsoft SQL server. In: EDBT/ICDT Workshop on Array Databases, Uppsala, Sweden (2011)
18. Szalay, A., Thakar, A.R., Gray, J.: The sqlLoader data-loading pipeline. *Comput. Sci. Eng.* **10**, 38 (2008)
19. Li, Y., et al.: A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence. *J. Turbul.* **9**(31), 1–29 (2008)
20. JHU IDIES Turbulence Database. <http://turbulence.pha.jhu.edu/>
21. McNutt, T., Nabhani, T., Szalay, A., Deweese, T., Wong, J.: Oncospace: EScience technology and opportunities for oncology. *Med. Phys.* **35**, 2900 (2008)

22. Szlavecz, K., et al.: Life under your feet: an end-to-end soil ecology sensor network, database, web server, and analysis service. Microsoft Technical report, MSR-TR-2006-90 (2006). <http://lifeunderyourfeet.org/>
23. Millennium Simulation Database. <http://gavo.mpa-garching.mpg.de/Millennium/>
24. Perlman, E., Burns, R., Li, Y., Meneveau, C.: Data exploration of turbulence simulations using a database cluster. In: Proceedings of the Supercomputing Conference (SC 2007) (2007)
25. Li, Y., et al.: A public turbulence database and applications to study Lagrangian evolution of velocity increments in turbulence. *J. Turbulence* **9**, N31 (2008)
26. Yu, H., et al.: Studying Lagrangian dynamics of turbulence using on-demand fluid particle tracking in a public turbulence database. *J. Turbulence* **13**, N12 (2012)
27. Eyink, G., et al.: Flux-freezing breakdown in high-conductivity magnetohydrodynamic turbulence. *Nature* **497**, 466 (2013)
28. Kanov, K., Burns, R., Eyink, G., Meneveau, C., Szalay, A.: Data-intensive spatial filtering in large numerical simulation datasets. In: The International Conference for High Performance Computing, Networking, Storage, and Analysis, Supercomputing SC 2012 (2012, submitted)
29. The Millennium XXL Project. http://wwwmpa.mpa-garching.mpg.de/mpa/research/current_research/hl2011-9/hl2011-9-en.html
30. Mitzenmacher, M.: A model for learned bloom filters and related structures. [arXiv:1802.00844](https://arxiv.org/abs/1802.00844) [cs.DS] (2018)
31. Kraska, T., Beutel, A., Chi, Ed.H., Dean, J., Polyzotis, N.: The case for learned index structures. [arXiv:1712.021208](https://arxiv.org/abs/1712.021208) [cs.DB] (2017)
32. Ortiz, J., Balazinska, M., Gehrke, J., Kehrt, S.S.: Learning state representations for query optimization with deep reinforcement learning. [arXiv:1803.08604](https://arxiv.org/abs/1803.08604) [cs.DB] (2018)



Spatio-Temporal Data Mining of Major European River and Mountain Names Reveals Their Near Eastern and African Origins

Peter Z. Revesz  

University of Nebraska-Lincoln, Lincoln, NE 68588, USA
revesz@cse.unl.edu

Abstract. This paper presents a spatio-temporal data mining regarding the origin of the names of the 218 longest European rivers. The study shows that 35.2% of these river names originate in the Near East and Southern Caucasus. The study also investigates the origin of European mountain names. It is shown that at least 26 mountain names originate from Africa.

Keywords: Data mining · Etymology · Mountain · River · Spatio-temporal

1 Introduction

Archeology reveals three main expansions of human populations into Europe. The first of these expansions is an expansion from North Africa that was likely prompted by the desertification of the Sahara. The second expansion took place as is a Neolithic agricultural expansion from Anatolia or perhaps even from Mesopotamia. The population of this expansion is often called the group of Early European Farmers (EEF). The third expansion is a Bronze Age nomadic expansion from the Eurasian Steppe areas. The third expansion is commonly associated with the expansion of Proto-Indo-European (PIE) language speaking populations [1]. These expansions and other examples of the spread of human populations can be studied today using various archaeogenetics methods [2–4]. However, neither archeology nor archaeogenetics can identify the languages of the early North Africans and the EEFs. The goal of this research is to identify the languages of these groups based on the old European river and mountain names that may derive from those languages.

The approach we take is to analyze ancient river and mountain names. Many of these topological names are presumed to have survived for millennia, that is, they reflect a pre-Indo-European era [5]. Therefore, these topological names can be associated with the native Ice Age Europeans, the early North Africans of the first expansion mentioned above, or the Neolithic EEFs.

This paper is organized as follows. Section 2 describes the data sources. Section 3 presents the results of data mining for the origin of the European river and mountain names. Section 4 discusses the linguistic implications of the data mining. The data mining implies ancient cross-continental linguistic connections between Africa and Europe. Finally, Sect. 6 presents some conclusions and ideas for future work.

2 Data Sources

2.1 River Names

Herodotus wrote extensively about the places he visited around 500 BCE. Strabo's work *Geographia* contains a wealth of river names that existed in the first century and was known to the ancient Greeks and Romans.

In a recent study of the degree of preservation of European river names, Carsten Peust [6] considered all European rivers exceeding 250 km that flow into the Atlantic, the Mediterranean Sea, the Black Sea or the Baltic Sea. He disregarded rivers that flow into the Arctic Sea and the Caspian Sea because supposedly the ancient Greek and Roman writers, such as Herodotus and Strabo, could not have known about those rivers. In this way, he obtained a total of 210 river names. Peust's goal was to find the degree that the names recorded by classical writers and used by modern populations match. He found a very high preservation rate, meaning that many river names seem to persist in Europe. Although Peust's study shows that many river names may be ancient, going back to the Neolithic or earlier, his study did not reveal the origin of these river names, which is the goal of our study.

We used his list with the addition of eight river names that we thought may be of ancient, non-PIE origin. Therefore, we used a total of 218 river names in our study. In particular, we added the following river names:

1. Aragón, this is a significant river flowing through the Basque region of Spain.
2. Arga, this is a significant river in Spain. The similar sounding Aragón river strengthens the proposition that this is an ancient river name.
3. Arnus, which just barely missed the arbitrary 250 km cut. It was the main Etruscan river.
4. Kama, which is a major river flowing into the Volga. This river flowed through the Finno-Ugric homeland according to the most widely-accepted theory of the origin of Finno-Ugric peoples.
5. Rioni, which flows into the Black Sea and is the longest river in Georgia. It seems that this barely missed Peust's criteria because Georgia is counted as a non-European state.
6. Salla, to whose length one may add the narrow and shallow lake Balaton (72 km in length) that can be considered a continuation of the Salla river. With that addition, it also just barely misses the arbitrary 250 km cut.
7. Saravus, which just barely missed the arbitrary 250 km cut.
8. Volga, which is the major river flowing into the Caspian Sea.

As a separate list, we created a list of ancient Near-Eastern river names. We placed into this list ancient Hattic river names from Anatolia because the Hattic people are assumed to have been indigenous to Anatolia before the arrival of PIE groups. We also added some Syrian, Caucasian and Mesopotamian river names.

2.2 Mountain Names

As the possible derivatives list, we considered all European mountain names from Herodotus, Strabo, and Pliny. We also considered all modern mountain names from the

Carpathian Basin, and area where many migrants passed through over the millennia, including the EEFs.

For the possible sources list, it seemed unwarranted to restrict attention only to the Fertile Crescent, which has few mountains, or to Anatolia, which is mostly a high plateau. Instead, we extended our search for mountain names to Africa. Since Africa is a huge continent with many mountains, hills and promontories, to cut down the search space, we considered what locations an ancient migrant population slowly migrating from North Africa into Europe would go through. Presumably, this ancient mountain-name-giver population would also name the mountains that it found along the way. This consideration of possibilities greatly cut down the search place, as explained below.

Migrants from North Africa into the Iberian Peninsula likely followed the path of the Strait of Gibraltar. In ancient times the North African side of the Strait of Gibraltar was called the Abile Mountain, which is recorded by Strabo.

Migrants from North Africa to the Italian Peninsula would have to pass the island of Sicily, where the largest mountain is called Nebrode Mountain, which is also recorded by Strabo. Although technically Sicily is part of Europe, this name could have been given to the mountain by one of the earliest African migrant groups that passed through the island.

Finally, migrants from North Africa could follow the Nile River and then move along the Eastern Mediterranean seashore into Anatolia and then Greece. On this way, they could have seen the Abydos Mountain, which was already recorded in pre-dynastic Egypt [7]. At the foot of this mountain is a city and a famous sanctuary.

Therefore, the possible sources list consisted of the following mountain names: Abile, Nebrode, and Abydos.

2.3 Congruent Sound Groups (CSGs)

First, we group some consonants together because they may change relatively easily from one to another. For these groups, we used the International Phonetic Alphabet's phonetic symbols and categorizations. The congruent sound groups (CSGs) are the following:

1. /b/, /p/, /m/ and /n/ – Here /b/ and /p/ is a voiced/voiceless plosive bilabial pair. These can change to /m/, which is a nasal bilabial sound. The /m/ can change to /n/, which is another nasal sound.
2. /d/, /t/ and /θ/ – Here /d/ and /t/ is a voiced/voiceless plosive dental/alveolar pair. The /θ/ is an aspiration of the /t/ sound.
3. /f/, /v/ and /w/ – Here /f/ and /v/ is a voiceless/voiced fricative labiodental pair. The /v/ and /w/ exchange is also common.
4. /g/, /k/ and /h/ – Here /g/ and /k/ is a voiced/voiceless plosive velar pair. The /k/ can change to /h/. For example, in reconstructed Proto-Finno-Ugric words with a word initial /k/ followed by a deep vowel the /k/ regularly changes to an /h/.
5. /l/ and /r/ – These alveolar sounds can commonly change into each other. Some ancient scripts, for example Mycenaean Linear B, did not even distinguish between these two sounds.
6. /s/, /z/, /ʃ/ and /ʒ/ – These fricative velar sounds can be commonly exchanged.

3 River Name Groups (RNGs)

River names can have a beginning of the form $V_1C_1V_2C_2$ where C_1 and C_2 are (not necessarily distinct) consonants and V_1 and V_2 are (not necessarily distinct) vowels or the empty string. We divided into separate groups all river names according to C_1 and C_2 . Table gives examples of Pre-Indo-European river names for each group for which we could find an appropriate match (Table 1).

Table 1. River name groups identified according to the first two consonants in the river name. The first consonant is in the row and the second consonant is in the column. Where we could find, we give a Pre-Indo-European river name. The legend of the superscripts is: As = Assyrian, Can = Canaanite, Cau = Caucasian, Ha = Hattic.

	b, p, m n	d, t, θ	f, v, w	g, k, h	l, r	s, z, ʃ, ʒ
b, p, m n					<i>Maraššantiya</i> ^{Ha}	
d, t, θ	<i>Adonis</i> ^{Can}					
f, v, w						
g, k, h	<i>Kummelmalya</i> ^{Ha}				<i>Hulaya</i> ^{Ha}	
l, r	<i>Arantu</i> ^{As}			<i>Araxes</i> ^{Cau}		
s, z, ʃ, ʒ	<i>Šamura</i> ^{Ha}				<i>Šariya</i> ^{Ha}	

3.1 Mountain Name Groups (MNGs)

The first consonant C_1 in each mountain name has to be /b/, /p/, /m/, or /n/. The second consonant C_2 has to be a /d/, /t/ or /θ/ in the Abydos group, or an /l/ in the Abile group. Finally, in the Ne-brode group (where Ne is ignored) the C_2 has to be an /l/ or an /r/ and the third consonant C_3 has to be /d/, /t/ or /θ/. Table 2 gives a summary of these three cases.

Table 2. River name groups identified according to the first two consonants in the river name. The first consonant is in the row and the second consonant is in the column. Where we could find, we give a Pre-Indo-European river name.

	$C_2 = d, t, \theta$	$C_2 = l$	$C_2 = l, r$ and $C_3 = d, t, \theta$
b, p, m n	<i>Abydos</i> ^{Egypt}	<i>Abile</i> ^{Morocco}	<i>Ne-brode</i> ^{Sicily}
other			

4 Analysis of the Spread of River and Mountain Names

4.1 Analysis of the Spread of River Names

Figure 1 shows the river names that fit the Adonis group. Altogether nine out of the 218 river names fit into this group.

Rank	Ancient Name	Modern Name	km
	Adonis (Canaanite)		
1	<i>Danuvius</i> (Latin)	<i>Donau</i> (Germ.), <i>Dunaj</i> (Slovak), <i>Duna</i> (Hung.), etc.	2860
3	<i>Tanais</i> (Greek)	<i>Don</i> (Russ.)	1950
91	<i>Tiberis</i> (Latin)	<i>Tevere</i> (Ital.)	410
98	<i>Tonzos</i> (Greek)	<i>Tunca</i> (Turk.)	390
113	<i>Tibiskos</i> (Greek)	<i>Temes</i> (Hung.), <i>Timiș</i> (Roman.)	360
125	<i>Tamesa</i> (Latin)	<i>Thames</i> (English)	350
168		<i>Dunajec</i> (Pol., Slovak)	290
188	<i>Tanarus</i> (Latin)	<i>Tanaro</i> (Ital.)	280
189		<i>Tomes</i> (Span.)	280

Fig. 1. Peust's river names that fit the Adonis River name group.

Rank	Ancient Name	Modern Name	km
	Hulaya (Hattic)		
36	<i>Hierasos</i> (Greek), <i>Gerasus</i> (Lat.)	<i>Sirét</i> (Roman.), <i>Seret</i> (Ukrain.)	710
40	<i>Garumna</i> (Latin)	<i>Garonne</i> (French)	650
45		<i>Glomma</i> (Norweg.)	600
72		<i>Gáláseatnu</i> (Sami), <i>Kalixälven</i> (Swedish)	450
133		<i>Körös</i> (Hung.), <i>Criș</i> (Roman.)	330
148		<i>Kalitvá</i> (Russ.)	310
154		<i>Xoról</i> (Russ., Ukrain.)	310
155	<i>Haliakmön</i> (Greek)	<i>Aliákmonas</i> (Greek)	300
158	<i>Granouas</i> (Greek)	<i>Hron</i> (Slovak), <i>Garam</i> (Hung.)	300
160	<i>Colapis</i> (Latin)	<i>Kolpa</i> (Sloven.), <i>Kupa</i> (Croat.)	300
170		<i>Hornád</i> (Slovak), <i>Hernád</i> (Hung.)	290
	Kummelmalya (Hattic)		
		<i>Kama</i> (Russia)	1,805
19	<i>(H)iberus</i> (Latin)	<i>Ebro</i> (Span.)	910
46		<i>Kemijoki</i> (Finland)	550
101	<i>Kanentelos</i> (Greek)	Charente (France)	380
110	Singilis (Latin)	<i>Genil</i> (Span.)	360
208	<i>Panysos</i> (Greek)	<i>Kámčija</i> (Bulgar.)	250
	Maraššantiya (Hattic)		
2	<i>Borysthenēs</i> (Greek)	Dnepr (Russ.), Dnipró (Ukrain.)	2290
15	<i>Parthiscus</i> (Latin)	Tisza (Hung.), Tisa (Serb.), Týsa (Ukrain.)	960
16	<i>Pyretos</i> (Greek)	<i>Prut</i> (Rom., Ukrain.)	950
29	<i>Marisos</i> (Greek)	<i>Maros</i> (Hung.), <i>Mureș</i> (Rom.)	770
30		<i>Prypeć</i> (Polish), <i>Príp'at'</i> (Russ.), <i>Prýp'jat'</i> (Ukrain.)	770
44		<i>Bereziná</i> (Russ.), <i>B'arézina</i> (Beloruss.)	610
54	<i>Ebros</i> (Greek)	<i>Marica</i> (Bulg.)	510
56		<i>Neris</i> (Lithuanian)	510
58	<i>Margos</i> (Greek)	<i>Mòrava</i> (Serb)	490
62		<i>Narew</i> (Polish), <i>Náraü</i> (Beloruss)	480
74		<i>Mura</i> (Croat, Hung. Slove.), <i>Mur</i> (Germ.)	450
120	<i>Marus</i> (Latin)	<i>March</i> (German), <i>Morava</i> (Czech/Slovak)	350
142		<i>Piřica</i> (Polish)	320
150		<i>Mulde</i> (Germ.)	310
166		<i>Bãrlád</i> (Roma.)	290
176		<i>Úbort'</i> (Russ., Ukrain.)	290
179		<i>Ibar</i> (Serb.)	280
185		<i>Pl'ússa</i> (Russ.)	280
194		<i>Polá</i> (Russ.)	270

Fig. 2. Peust's river names that fit the Hulaya, Kummelmalya and Maraššantiya river name groups.

Rank	Ancient Name	Modern Name	km
	Šamura (Hattic)		
13	Savus (Latin)	Sáva (Croat; Serb), Sava (Sloven.)	990
79		San (Pol.), S'an (Ukrain.)	440
93	Samus (Latin)	Szamos (Hung.), Sómeş (Roman.)	400
97	Sēnos (Greek)	Shannon (Engl.), an tSiannain (Irish)	390
124	Sabrina (Latin)	Severn (Engl.), Hafren (Welsh)	350
143		Samára (Russ., Ukrain.)	320
151	Asamus (Latin)	Osem (Bulgar.)	310
175		Savalá (Russ.)	290
187		Suunujoki (Finn.), Súna (Russ.)	280
204	Samara (Latin)	Somme (French)	260
	Šariya, Zuliya (Hattic)		
27		Sal (Russ.)	800
75		Sluč' (Russ.), Sluč (Ukrain.)	450
88	Salas (Greek)	Saale (German)	410
112		Súla (Russ.), Sulá (Ukrain.)	360
132		Zsil (Hung.), Jiu (Rom.)	330
152		Sarthe (French)	310
159	Isara (Greek)	Isar (German)	300
	Saravus (Latin)	Sarre (French), Saar (German)	246
	Salla (Latin)	Zala (Hung.)	139
	Arantu (Assyrian)	Orontes (Greek), Asi (Arabic, Turkish)	
5	Rhenus (Latin)	Rhein (Germ.), Rhin (French), Rijn (Dutch)	1230
7	Albis (Latin)	Elbe	1090
92		Ljungan (Swed.) < * Lungan	400
	Phasis (Greek)	Rioni (Georgia)	333
173		Łyna (Pol.), Alna (Lith.), Láwa (Russ.), Alle (Germ.)	290
182		Leine (Germ.)	280
186	Arrabo (Latin)	Raab (Germ.), Rába (Hung.)	280
	Arnus (Latin)	Arno (Italy)	241
	Araxes (Caucasian)	Arax (Armen., Georgian), Araz (Azer.)	1,072
116		Árgeş (Roman.)	350
119		Lúga (Russ.), Laugaz (Votik)	350
178	Argianēs (Greek)	Ergenes (Turk.)	280
		Arga (Spain)	145
		Aragón (Spain)	129

Fig. 3. Peust's river names that fit the Šamura, Šariya, Arantu and Araxes river name groups.

Figure 2 shows the river names that fit the Hulaya, Kummelmalya and Maraššantiya river name groups. Figure 3 shows the river names that fit the Šamura, Šariya, and Arantu and Araxes river name groups.

In Table 3 shows the number of river names from our list of 218 rivers that belong to each RNG that is associated with a Near Eastern river name.

Table 3. The number of river names that fall within each river name group.

	b, p, m n	d, t, θ	f, v, w	g, k, h	l, r	s, z, ʃ, ʒ
b, p, m n					19	
d, t, θ	9					
f, v, w						
g, k, h	6				11	
l, r	8			5		
s, z, ʃ, ʒ	10					9
Total	33			5	39	

Table 3 shows that 77 European major river names out of 218, that is, 35.3%, could be traced to the Near East and the Caucasus. That is an unexpectedly high number given that some European river names could also derive from the native Ice Age hunter-gatherers, the early North Africans, or from the Bronze Age Proto-Indo-Europeans. Therefore, this is a strong linguistic support to the thesis that a large number of European river names go back to the agricultural expansion that started c. 7000 BC.

It is also interesting that the distribution of the river names is not random. There is a clear preference for the second consonant to be either in the first group (*b, p, m* and *n*) or the fifth group (*l* and *r*). Moreover, all of these names are associated with the Near East, and a remarkably large number with the Hattic names. Hence the mentioned preference probably reflects an essential characteristic of the Hattic language, who may have been the earliest agriculturalists in Anatolia. They apparently spread over the entire European continent and left their marks in all the places that was the main route of expansion, including present day Bulgaria, Romania, Serbia, Croatia, Slovenia, Hungary, Slovakia, Poland, Lithuania, Ukraine, Russia, Austria, Germany, Netherlands, Norway, Sweden, Finland, England, Ireland, France, Italy and Spain.

When the second syllable was in the fourth group (*g, k* and *h*), which was associated with the Araxes River in the Caucasus, then the distribution of the names was more selective. The five names in this group can be divided into a one subgroup, which starts with a vowel and the first consonant is an *r*, while the second subgroup starts with an *l* and contains the name of only one river near St. Petersburg, Russia.

The first subgroup apparently had a maritime expansion from the Caucasus to the historical Thrace (approximately present day southern Romania, Bulgaria and Northern Greece) then it may have reached the Argolis plain in Greece, and then reached northern Italy, where there is an Orco River, which is a tributary of the Po River, then reached southern France, where there is an Ariège River, which is a tributary of the Garonne River, and finally it reached northern Spain. The Loire River, which originates in southern France, also has the Arroux River as a tributary. It is possible that this group of people called the entire Po the Orco and the entire Loire the Arroux, but these latter names survive today only as the names of tributaries.

The second subgroup probably had a separate, unrelated development because of the combination of the geographic and the subtler linguistic differences. The only route connecting the Caucasus with the northeast of Russia would be via the Volga River.

The ancient name of Volga was the Scythian Rā, which has /r/ as the first consonant but has no second consonant. Therefore, whether this indicates an expansion from the Caucasus or not cannot be decided by this study.

Record	Ancient Name	Modern Name
Strabo	Abile (Morocco)	
Strabo	Am- pe los (Samos I., Greece)	
Strabo	Orbe los (Thracian)	
Strabo	Pe lion (Thessaly, Greece)	
Strabo	Pho loe. (West Greece)	* Poloe
		Bilo (Hu/Croat.)
		Béli (Hu/Rom.)
Strabo	Nebrode (Sicily)	
Strabo	Bertiscos (Thessaly, Greece)	
Strabo	Parthenion (Athens, Greece)	
Strabo	Platamodes prom. (Greece)	
Strabo	Typhrestos (Thessaly, Greece)	* Ty-prettos
pre-Dynastic	Abydos (Egypt)	
Pliny	Car- pat (Central Europe)	
Strabo	Emodos (India)	
Strabo	Hy- mettos (Athens, Greece)	
Strabo	Hy- paton (Greece)	
Strabo	Idu- beda (Iberia)	
Strabo	Lyka- bettos (Athens, Greece)	
Strabo	Oros- peda (Iberia)	
Strabo	Pedalion prom. (Cyprus)	
Strabo	Ptoon (Greece)	
Strabo	Pydna (Greece)	
Strabo	Sar- pedon prom. (Cilicia)	
		Bada -csony (Hung.)
		Bodoki (Hung., Roman.)
		Buda (Hung. city and hills)
		Buda -hunga (Burundi)
		Fátra (Hung., Slovak) < * Pátra
		Madaras (Hung., Slovak)
		Pádis plateau (Hung., Roman.)
		Págyes prom. (Hung., Roman.) < * Pádes

Fig. 4. Mountain names that fit the three mountain name groups.

4.2 Analysis of the Spread of Mountain Names

Figure 4 shows the list of mountain names that were found to fit well with the three source names. Naturally, the list contains a large number of Greek mountain names because that area was the most familiar to Strabo. Although the study of Asian

mountain name was not the goal of this study, we also added to the Abydos list in Fig. 4 the famous Emodos Mountain, which today is called the Himalayas Mountain, the Sarpedon promontory in Cilicia, which is in southern Turkey, and the Pedalion promontory on Cyprus. It is possible that some of the ancient African migrants instead of following the Eastern Mediterranean coast followed a different route into India, probably going around the Arabian Peninsula. The Burundi mountain name Buda suggests that the expansion of the population that gave these mountain names started from around Lake Victoria, which is a source of the Nile River.

In summary, we found 6 European mountain names in the first group, 5 European mountain names in the second group (including Nebrode), and 15 mountain names in the third group. Therefore, we found a total of 26 mountain names that seem to derive from African sources. We also found a Burundian mountain name supporting the hypothesis of a mountain name giving population expansion along the Nile River.

5 Discussion of the Linguistic Implications

5.1 River Name Etymologies

The Araxes river name group (especially its first subgroup) recalls the *Argonauts* in Greek mythology. These mythical heroes made a journey to the land of the Colchis, which is present day Georgia. This myth may reflect some knowledge about the expansion of this first subgroup from the Caucasus to Greece. There are some linguistic theories that connect the Kartvelian languages, which includes Georgian, to the Basques [8]. If the connection is valid, then the maritime expansion of this group may be the explanation for the linguistic connection.

The etymology of the Arantu river is unknown. It may be a Hurrian name in origin. The Arnus is the main Etruscan river. There are some indications of Etruscan-Hurrian language relationships.

The word Maraššantiya may be related to puro^{Finnish} (creek) and folyó^{Hungarian} (river). Another surprising finding is that the Hattic river names all end in -ya. This could be related to jő^{Hungarian} (river), which is the last syllable of many Hungarian river names.

5.2 Mountain Name Etymologies

It already struck us that the three mountain name groups Abile, Nebrode, and Abydos may be cognates. That is, they may all derive from some common root, some African word of truly ancient origin that may be Proto-Word for many languages in the world.

For example, Abydos and Bile may be related because a /d/ to /l/ change is not unusual. Hence *Abydo > *Abilo > Abile is a possible development. In addition, the insertion or deletion of an /r/ is also possible. Moreover, Nebrode could be a composite word consisting of Ne + Abrode. We also could have with an /r/ omission the following development: *Abrode > *Abode > Abydo. Therefore, it is not impossible that one ancient North African population split and migrated on separate route into Europe. During this migration, the original word for “mountain” became slightly modified.

To further investigate this possibility, we searched for possible cognate words in various African, Eurasian languages. Table 4 lists the words that we could identify as possible cognates. Note that the Somali word *buurta*, which means “hill” could be close to the proto-word we look for. Our guess is that the proto-word was **aburda*. The Somali language is a distinct branch of the Afro-Asiatic languages, but the proto-word **aburda* likely pre-dates the beginning of Proto-Afro-Asiatic because it is also related to Sumerian *bàd*, which means “wall.” The Sumerian wall is clearly cognate because walls are made of rocks and bricks. The proto-word is also related to the Filipino *bato*, which means “rock.”

It is also surprising that the Carpathian Mountains now can be given a Sumerian etymology. In Sumerian *kur* means “mountain” and combines with *bàd*, it means mountaintop [9]. Previously, no satisfying etymology was given for the name Carpat, which was already recorded by Pliny.

Table 4. Some words that are cognates of various mountain name groups.

Group		Definition	Source
<i>Abile</i>	<i>pe^{lla}</i> ^{Greek}	Stone, rock	[10]
<i>Abile</i>	<i>pa^{alu}</i> ^{Finnish}	Pile	[13]
<i>Abydos</i>	<i>apa^{ta}</i> ^{Yoruba}	Rock	[13]
<i>Abydos</i>	<i>ba^{to}</i> ^{Filipino}	Rock	[13]
<i>Abydos</i>	<i>b^{àd}</i> ^{Sumerian}	Wall	[9]
<i>Car-pat</i>	<i>ku^r-ba^d</i> ^{Sumerian}	Mountaintop	[14]
<i>Abydos</i>	<i>buda</i> ^{Hungarian}	Sharp picket	[11]
<i>Abydos</i>	<i>pa^{ti}</i> ^{Mansi}	Go down to the river from a mountain	[12]
<i>Abydos</i>	<i>bud^{ja}</i> ^{Proto-Albanian}	Lip, end, edge, bank, stitch, rock	[10]
<i>Abydos</i>	<i>py^{tna}</i> ^{Greek}	Stone, hill rock	[10]
<i>Abydos</i>	<i>pe^{tra}</i> ^{Greek}	Stone, rock	[10]
<i>Ne-brode</i>	<i>bla^t</i> ^{Maltese}	Rock	[13]
<i>Ne-brode</i>	<i>bu^{urta}</i> ^{Somalian}	Hill	[13]

Previous research of the author [15–19] using phylogenetic and spatio-temporal data mining methods [20, 21] has revealed a close language connections between Minoan, Hattic and Hungarian. This result seemed to contradict previous theories about the origin of the Hungarian people and language. However, Fig. 5 suggests an explanation for the relationships. The white circles in Fig. 5 show the geographic locations that may provide refuge areas against any nomadic invasion from the great Eurasia Steppe areas. It can be assumed that in general the nomadic invaders, be they Bronze Age Proto-Indo-Europeans [1] or later groups, would have preferred the relatively flat areas north of the Carpathian Mountains than to go through the Carpathians with their horses. They would also avoid the northern forest areas, which may be a refuge area other Uralic people. Similarly the Caucasus Mountains, Apennine Mountains and the Pyrenees Mountains may have provided some refuge for the Caucasian, the Etruscan and the Basque populations, respectively.

However, while all the white circles are shielded from an invasion from the Eurasian Steppe, they fare differently from an invasion from Anatolia. The Basque and the Etruscan areas would be still shielded, but most of the Carpathian Basin would be exposed to a southern invasion along the Danube River. It is possible perhaps that the relatively mountainous areas of Transylvania would have some protections, but that protection would be much weaker than the protection provided by the Carpathian Mountains against an eastern invasion. That means that a Hattic expansion could have influenced the population and culture of the Carpathian Basin, but the Proto-Indo-European invasion could have only a minimal impact.

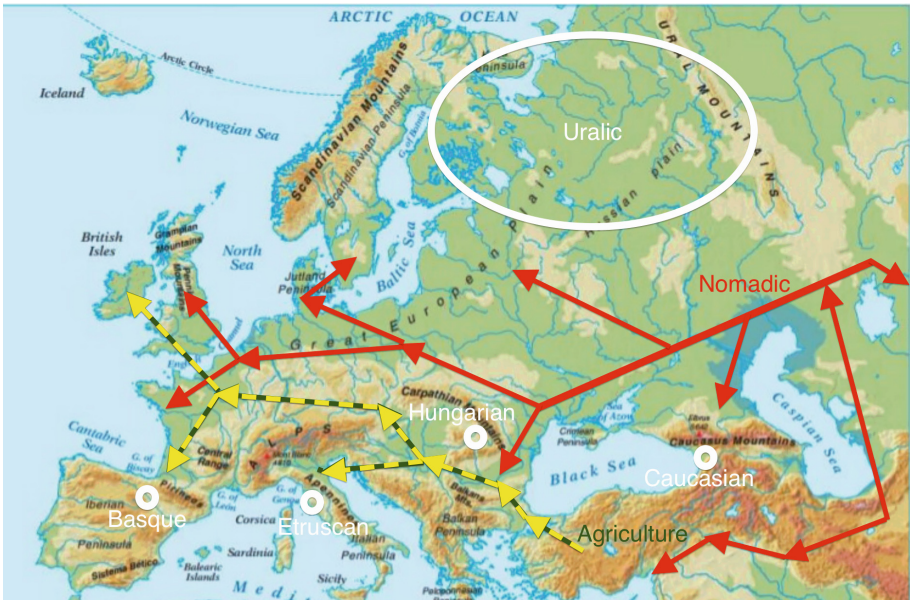


Fig. 5. This figure shows two main expansions of human populations into Europe: (1) an agricultural expansion from Anatolia (dashed yellow lines) and (2) a nomadic expansion from the Eurasian steppe areas solid red lines). The white circles indicate possible language refuge areas. (Color figure online)

Figure 5 also suggests that the Proto-Indo-Europeans may have come to Europe from two directions. The first direction would have been from the Eurasian Steppe and the second via Anatolia. The second direction is a rougher terrain, which would have greatly slowed down the speed of the Proto-Indo-European advance. Moreover, in the Aegean Sea islands, as well as Sicily and Sardinia, the earlier populations would have avoided an immediate invasion and could have their culture survive longer than on the mainland of Europe. This seems to be especially the case in the Cyclades and in Crete. The island of Crete was the seat of the Minoan civilization. The Minoan civilization seems related to the Hattic civilization, but survived much longer. Hattusa, the center of Hattic civilization, fell to the Hittites around 1700 BCE. However, the Mycenaean

Greeks only captured Crete around 1450 BCE. According to many historians, the Mycenaean conquest would have been further delayed if the eruption of a volcano on the island of Santorini would not have already weakened by that time the Minoan civilization.

6 Conclusions and Further Work

Our spatio-temporal data mining study considered the geographic distribution of river and mountain names over a period of several millennia. We found evidence of an African origin of many mountain names in Europe, and even in Asia. Our hypothesis is that these mountain names were brought into Europe when the Sahara dried up and hunting-gathering North Africans were seeking a better climate. This gradual desertification may have started at the end of the Ice Age. Hence probably these mountain names could go back to the Ice Age.

We also found evidence of a Near Eastern or Caucasian origin of many river names. Apparently the EEFs either did not know the names of the rivers, or they found it very important to call the rivers certain names. On the other EEFs seem to have been less interested in the mountains and were willing to accept for them the local names that the hunter-gatherers already gave them.

One interesting question that remains is whether these river and mountain names extend to Australia and the Americas. If the river and mountain names have cognates in these continents, then that fact would imply an even earlier origin of these names, going back perhaps to the very earliest human proto-language.

References

1. Anthony, D.: Horse, wagon and chariot: Indo-European languages and archaeology. *Antiquity* **69**(264), 554–565 (1995)
2. Revesz, P.Z.: A mitochondrial DNA-based model of the spread of human populations. *Int. J. Biol. Biomed. Eng.* **10**, 124–133 (2016)
3. Revesz, P.Z.: A last genetic contact tree generation algorithm for a set of human populations. In: 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, pp. 501–502. ACM Press, New York (2016)
4. Revesz, P.Z.: A spatio-temporal analysis of mitochondrial DNA haplogroup I. In: 20th International Conference on Circuits, Systems, Communications and Computers, MATEC Web of Conferences, vol. 76, no. 04048 (2016)
5. Krahe, H.: *Unsere ältesten Flussnamen*. Otto Harrassowitz, Wiesbaden (1964)
6. Peust, C.: How old are the river names in Europe? A glottochronological approach. *Linguistik Online* **70**(1) (2015)
7. Wegner, J.: From elephant-mountain to anubis-mountain how old are the river names in Europe? A theory on the origins and development of the name Abdju. Glottochronological approach. *The Archaeology and Art of Ancient Egypt. Essays in Honor of David B. O'Connor II*, *Annales du Service des Antiquités de l’Égypte, Cahiers*, vol. 36, pp. 459–476 (2007)

8. Sturua, N.: On the Basque-Caucasian hypothesis. *Studia Linguistica* **45**(1–2), 164–175 (1991)
9. Halloran, J.A.: *Sumerian Lexicon*. Logogram Publishing, Los Angeles (2006)
10. Palaeolexicon Homepage. <https://www.palaeolexicon.com>. Accessed 1 June 2018
11. Czuczor, G., Fogarasi, J.: *A magyar nyelv szótára*. vol. 1, Athenaeum, Budapest (1862). http://mek.oszk.hu/05800/05887/pdf/1kotet_2.pdf. Accessed 1 June 2018
12. Mansi Dictionary of Munkácsi and Kálmán Homepage. <http://www.babel.gwi.uni-muenchen.de/munka/index.php>. Accessed 1 June 2018
13. Google Translate Homepage. <https://translate.google.com>. Accessed 1 June 2018
14. The Pennsylvania Sumerian Dictionary Homepage. <http://psd.museum.upenn.edu/epsd1/nepsd-frame.html>. Accessed 1 June 2018
15. Revesz, P.Z.: Bioinformatics evolutionary tree algorithms reveal the history of the Cretan Script Family. *Int. J. Appl. Math. Inform.* **10**(1), 67–76 (2016)
16. Revesz, P.Z.: A computer-aided translation of the Phaistos Disk. *Int. J. Comput.* **10**(1), 94–100 (2016)
17. Revesz, P.Z.: A computer-aided translation of the Cretan Hieroglyph script. *Int. J. Sig. Process.* **1**, 127–133 (2016)
18. Revesz, P.Z.: A translation of the Arkalochori Axe and the Malia Altar Stone. *WSEAS Trans. Inf. Sci. Appl.* **14**(1), 124–133 (2017)
19. Revesz, P.Z.: Establishing the West-Ugric language family with Minoan, Hattic and Hungarian by a decipherment of Linear A. *WSEAS Trans. Inf. Sci. Appl.* **14**(1), 306–335 (2017)
20. Revesz, P.Z., Wu, S.: Spatiotemporal reasoning about epidemiological data. *Artif. Intell. Med.* **38**(2), 157–170 (2006)
21. Revesz, P.Z.: *Introduction to Databases: From Biological to Spatio-Temporal*. Springer, New York (2010). <http://dx.doi.org/10.1007/978-1-84996-095-3>

Information Extraction and Integration



Query Rewriting for Heterogeneous Data Lakes

Rihan Hai¹(✉), Christoph Quix^{1,2}, and Chen Zhou¹

¹ Databases and Information Systems, RWTH Aachen University, Aachen, Germany
{hai, zhou}@dbis.rwth-aachen.de

² Fraunhofer-Institute for Applied Information Technology FIT,
Sankt Augustin, Germany
christoph.quix@fit.fraunhofer.de

Abstract. The increasing popularity of NoSQL systems has led to the model of polyglot persistence, in which several data management systems with different data models are used. Data lakes realize the polyglot persistence model by collecting data from various sources, by storing the data in its original structure, and by providing the datasets for querying and analysis. Thus, one of the key tasks of data lakes is to provide a unified querying interface, which is able to rewrite queries expressed in a general data model into a union of queries for data sources spanning heterogeneous data stores. To address this challenge, we propose a novel framework for query rewriting that combines logical methods for data integration based on declarative mappings with a scalable big data query processing system (i.e., Apache Spark) to efficiently execute the rewritten queries and to reconcile the query results into an integrated dataset. Because of the diversity of NoSQL systems, our approach is based on a flexible and extensible architecture that currently supports the major data structures such as relational data, semi-structured data (e.g., JSON, XML), and graphs. We show the applicability of our query rewriting engine with six real world datasets and demonstrate its scalability using an artificial data integration scenario with multiple storage systems.

1 Introduction

Data integration is an open challenge that has been addressed for decades; with the increasing diversity in the data management landscape the challenge has become even harder. One of the first problems in data integration is getting access to the source data. *Data lakes* (DLs) have been proposed to tackle the problem of data access by providing a comprehensive data repository in which the raw data from heterogeneous sources will be ingested in its original format [6, 15]. As DLs do not provide a common unified schema for the loaded data (as in data warehouses), the extraction of metadata from data sources is a crucial element in DLs [8, 13]. DL is still a relatively new concept; thus, there is still a discussion about the concrete functionalities and architectures. However, there are certain aspects that are generally agreed upon, e.g., DLs should provide a common interface to query integrated data across heterogeneous data sources [6, 14, 15].

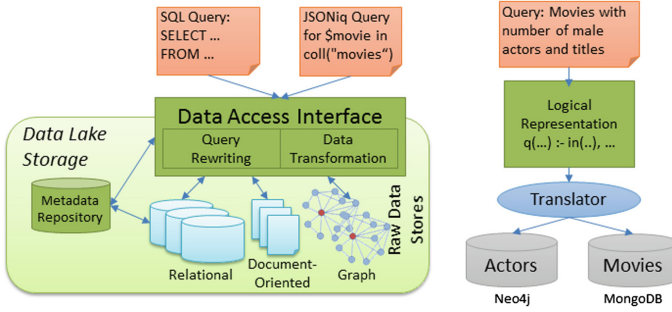


Fig. 1. Data lake storage layer and motivating example

In this work, we focus on the problem of query rewriting for *logical data lakes*, making use of a (partially) integrated schema and logical mappings between the sources and the integrated schema [14]. Figure 1 sketches the storage layer of a DL architecture [8] which provides a unified data access interface to heterogeneous raw data stores. A metadata repository maintains the schema information and the mappings to an integrated schema. To illustrate the challenge addressed in this paper, the right part of the figure sketches a motivating example (the formal representation of this example will be done in Sect. 2). Suppose we have two data stores, a graph database (Neo4j) with information about actors and a document-oriented database (MongoDB) with information about movies and their cast. To answer the query ‘movies with number of male actors and titles’, we have to ‘join’ the information from both data stores. For this, our system will first translate the input queries (which can be in different query formats, e.g., SQL or JSONiq [4]) into a common logical representation based on Datalog, which is independent of the underlying systems.

Several big data systems like Apache Spark are already able to access data in different types of DBMS; yet, the challenge which we address in this paper is that data for answering a single query is stored in several DBMS of various types. In addition, we also use schema mappings to perform a logical data integration, i.e., resolving the heterogeneities which are caused by different schema structures and semantics (e.g., different labels for types and attributes). Furthermore, our approach also handles queries that cannot be answered just by a union of the sources, i.e., our system performs additional query processing (e.g., join of source data) that cannot be handled by any system separately.

Our system SQRE (Scalable Query Rewriting Engine) executes the generated subqueries on each data store, retrieves the query results, executes the parts of the query not covered by a source, and creates the final integrated results. We use Apache Spark to execute the queries as it provides many connectors to various DBMS and enables with its Dataset abstraction an easy integration of the query results. We also push down selection predicates to the data sources to optimize query execution and to reduce the amount of data that has to be loaded.

The problem of processing queries that span multiple heterogeneous data stores and integrating the query results has been addressed recently in multistore (or polystore) systems [3, 7, 9]. These systems usually focus on the optimization and the efficient execution of queries spanning multiple systems; in contrast, our logical approach focuses on the heterogeneities caused by different query and schema languages as well as different schema structures; logical optimization is only a side effect of the approach. Especially, we study query rewriting for queries expressed in JSONiq, a query language for JSON data, which is becoming *the* data format for NoSQL systems and data exchange settings.

Our main contributions in this paper are as follows:

- We propose a query rewriting engine, which supports unified query interface over heterogeneous data stores in data lakes.
- We define a system-independent, logical query language to identify relevant data stores based on the logical schema mappings.
- We show experimentally the practicality, scalability, and efficiency of our proposed system over two use cases covering six real world data sets.

The remainder of the paper is organized as follows. In Sect. 2, we present first the overall architecture and then discuss in detail the major components as well as the logical approach for query rewriting and data integration of our system. The evaluation results are reported in Sect. 3, before we introduce related works in Sect. 4 and conclude the paper in Sect. 5.

2 Query Rewriting with SQRE

Figure 2 depicts the architecture of SQRE. The input query is parsed by the *Query Parser*, which creates the logical representation. The *Logical Form Analyzer* (LFA) takes this formula and creates an internal data structure as input for the *Predicate Filters* of the sources. These filters match the predicates with the schemas that are available in the sources. For this, source and the integrated schemas and their mappings have to be taken into account, as provided the meta-data repository in our data lake [6]. The output of the LFA and the predicate filters is a list of predicates that can be processed by each data store. The predicate list is then transformed by the *Translator* components to the query language of the corresponding data store, e.g., SQL for MySQL, Cypher for Neo4j. The executors will take the transformed queries, execute them, and return the result back to the *Data Source Integrator*. Finally, the *Data Transformer* will create the desired result structure.

The components of SQRE will be described in the subsequent sections with the running example: a data set for actors stored in Neo4j with schema as $S1: movie_actors(actor_id, name, gender)$; the second data set stored in MongoDB that describes the cast information with schema $S2: cast_info(actor_id, movie_id, movie_title)$. The integrated schema is $T: movie(movie_id, movie_title, actors(actor_id, name, gender))$. To keep the example simple, we removed some of the details, e.g., non-matching IDs, complex JSON structures. The schema

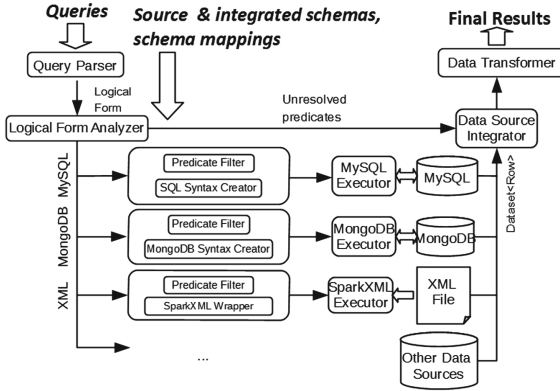


Fig. 2. Overall architecture of SQRE

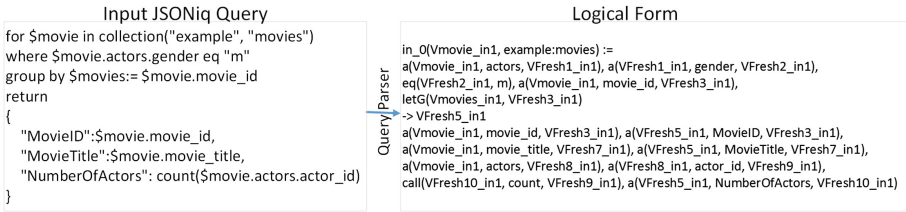


Fig. 3. Running Example: JSONiq Query and Corresponding Logical Rule

mappings are also simple in this example as they are element-to-element correspondences between elements with the same in the source schemas S_1 , S_2 , and the integrated schema T . More complex mappings for nested structures using Skolem functions are also supported (see below). Figure 3 shows the input JSONiq query, which returns the ID, title and total number of male actors for each movie in the collection *movies* (*example* is database name). We explain how SQRE parses, analyzes, and rewrites the given query in what follows.

2.1 Logical Representation of Queries

To provide sufficient extensibility for different query languages and flexibility for handling schema mappings, we use a logical representation based on Datalog (see Table 1) with a semi-structured data model. Figure 3 presents the input JSONiq query and its corresponding logical rule. The logical rule is divided into three parts: **head** := **body** -> **result**. The head defines the main variables of the query, the body is a conjunction of predicates that must be satisfied and the result part determines the structure of the result objects (these can also be nested) as specified in the return clause of the JSONiq query.

During the translation process, the variables of the original query are replaced with generated variables in the logical rule, e.g., *movie* is replaced with *Vmovie_in1*. The variables *VFresh1_in1* and *VFresh2_in1* represent the inter-

Table 1. An overview of predicates in the Query Parser

Predicate	Description
$in.i(x, c)$	The item x is a member of c (sequence or array), and i is the sequence number of c in the current logical form
$a(x, k, v)$	The item x is an object with a key k , and the value is v
$let(x, y)$	Assert y is the value of x
$letG(k, y)$	x is a group-by-key, whose value is y
$eq/ne/lt/le/gt/ge(x, y)$	$x = y/x \neq y/x \leq y/x < y/x \geq y/x > y$
$call(x, f, [a_1, \dots])$	x is the result of function f with arguments a_1, \dots

mediate objects in the path expression `movie.actors.gender`, `VFresh3_in1` is the movie ID. Since the query contains a group-by statement, we use `letG(Vmovies_in1, VFresh3_in1)` to state that the results should be grouped by `movie.movie_id` (variable `VFresh3_in1`). In the result part of the rule (after `->`), the return object is constructed. The main variable is `VFresh5_in1`, all attributes of the result (`MovieID`, `MovieTitle`, and `NumberOfActors`) are assigned with an a -predicate to this variable. Since there is an aggregation function `count` in the input query, it is specified with a `call` clause.

The **Logical Form Analyzer** (LFA) discovers equivalent predicates and tracks back to all related predicates to clarify the references of variables. Moreover, it analyzes the inner relations of predicates and transforms them into more meaningful internal representations (implemented as Java classes). In Fig. 3, the LFA first recognizes that the final result is represented as the variable after `->`, i.e., `VFresh5_in1`. Then, the LFA identifies all a -predicates whose first variable is also `VFresh5_in1`. In this case, it finds `a(VFresh5_in1, MovieID, VFresh3_in1)`. Next, in order to find the meaning of a variable in the original query, the LFA continues to track back the relevant predicate `a(Vmovie_in1, movie_id, VFresh3_in1)`. Finally, it reassembles `movie.movie_id` as the returned attribute path under the name of `MovieID`. The rest of the logical form is similarly analyzed, the result is shown in the upper part of Fig. 4.

JSONiq allows also nested FLWOR expressions, which will be translated to multiple rules. The LFA also supports analyzing nested JSONiq queries by traversing their subqueries and tracking back the related predicates in terms of variables. Not all JSONiq queries can be translated to the logical form, as it can represent only disjunctions of conjunctive queries.

2.2 Generation of Data Source Queries

To generate executable subqueries for the underlying DBMS, the Translators perform the following operations: (1) transform the internal representation from

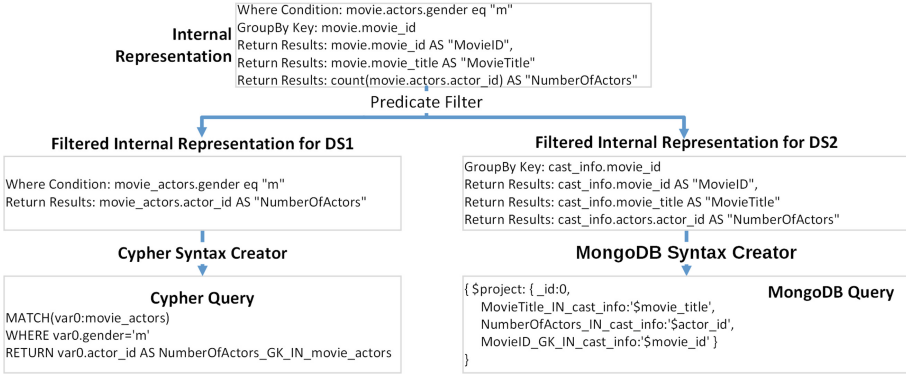


Fig. 4. Running example: from internal representation to executable queries

integrated schema to source schemas using the mappings; (2) validate explicit equality joins spanning multiple sources and add implicit join conditions; (3) generate executable queries for various data source types. SQRE instantiates for each data source a corresponding Translator with two subcomponents. The *Data Source Predicate Filters* handle the first two operations, while *Syntax Creators* rearrange the transformed predicates and generate executable subqueries.

The **Data Source Predicate Filters** replace the attribute names in the internal form with the attribute names in the current source based on the correspondences in the mapping. For example, in Fig. 4 for *DS1* the Predicate Filter replaces `movie.actors.gender` and `movie.actors.actor_id` in the internal representation with `movie_actors.gender` and `movie_actors.actor_id` (similarly for *DS2*). To guarantee the correctness of the rewritten query, there are two special cases in which the predicate filter adds new ReturnedResults to the list of internal representations for the current source. The first case applies to schema mappings referring to attributes with corresponding elements in multiple sources, e.g., `movie.actors.actor_id` in the integrated schema has corresponding schema elements in both *DS1* and *DS2*. Thus, the predicate filter adds an implicit join between the `actor_ids` of both sources. In the second case, if a source attribute corresponds to group-by-keys, this attribute is added to the list of returned variables.

Moreover, the filter also performs data type conversion in case that the types differ in the integrated schema and the sources. SQRE also supports the Skolem functions in nested GLAV mappings [16]. Skolem functions are applied for the grouping to create complex elements in nested relations. The predicate filter checks also which predicates can only be applied after the data has been merged from multiples sources (referred to as unresolvable predicates), e.g., joins, aggregation functions, and group-by clauses. These predicates are handled later by the Data Source Integrator.

In order to translate the internal form into corresponding queries for specific DB systems, the **Syntax Creators** are applied. A Syntax Creator translates also the function calls into corresponding functions or query expressions of the source systems. This is of course limited only to ‘known’ functions (e.g., count, max, contains) and cannot be applied to general user-defined functions. For example, the function `contains(a, "b")` (substring match), the Syntax Creator creates a *like* clause for SQL and Neo4j, and `{match {$regex: }}` for MongoDB. Figure 4 shows the results for the Neo4j and MongoDB queries. Notably, we use `_GK_` to mark the group-by-keys, e.g., `NumberOfActors_GK_IN_movie_actors`.

2.3 Query Execution on Data Sources

The Data Source Executors take the generated queries and execute them on the corresponding DBMS. For the MongoDB and Neo4j executors, we use Spark-Mongo-Connector¹ and Spark-Neo4j-Connector² to connect to databases and execute native queries. The XML executor uses Spark-XML³, while relational databases are accessed with the built-in JDBC interface of Spark. For the running example in Fig. 4, SQRE initializes a Neo4j Executor and a MongoDB Executor to execute the queries. The query results from the sources are transformed into structured *Datasets* with *Rows*, and passed to the Data Source Integrator.

2.4 Data Integration and Final Transformation

The results retrieved from each data source are not yet integrated, i.e., the unresolved predicates still have to be applied, e.g., group by, join, aggregation functions, and having clauses. In the running example, these predicates include the group-by-key `movie_id` and the count function on `actor_id`. To obtain the integrated results, the **Data Source Integrator** collects the unresolved predicates and forms a final SparkSQL query. Then, it executes the final SparkSQL query and stores the result in a dataset. There are two practical issues that need to be considered in this step. First, a join condition can either be defined explicitly in a query, or implicitly from results of schema matching. In the running example `actor_id` has corresponding attributes in both *DS1* and *DS2*. In SQRE, the integrator adds a *full outer join* clause based on such correspondences to integrate results from distinct sources, while join conditions in the original query are translated to an inner join. Second, as shown in Fig. 5 the Integrator merges matched columns in distinct sources into one column, since they are treated as one attribute in the integrated schema. SQRE applies a series of user defined functions (UDFs) based on SparkSQL to resolve the attribute name and type conflicts of these matched columns.

As JSONiq queries can be the input, the final result of the query needs to be transformed into JSON. In the previous step, Spark Datasets are created as an intermediate result, which correspond to relational tables. The **Data**

¹ <https://docs.mongodb.com/spark-connector/v1.1/java-api/>.

² <https://github.com/neo4j-contrib/neo4j-spark-connector>.

³ <https://github.com/databricks/spark-xml>.



Fig. 5. Running example: the final spark query and final query results

Transformer transforms the datasets into a JSON structure using the built-in methods of Spark. In the case that the query groups the result by a key and returns a set of elements with the same key in a single JSON object, we perform an additional step using user defined aggregation functions (UDAFs).

3 System Evaluation

We have evaluated the performance of SQRE over six real world data sets. First, we compare SQRE with a baseline approach, which transforms data stored in MongoDB and Neo4j to MySQL, and executes queries only in MySQL. We show that our approach significantly cuts down the total query processing time (including the time for data transformation) compared to the baseline approach. Furthermore, we design three data integration scenarios with queries, and demonstrate that our query rewriting approach scales well to the increase of query complexity. The experiments in this section are performed on a Intel i7 2.6 GHz machine with 12 GB RAM. We have implemented our framework in Java and used Apache Spark 2.2.1. We have run the experiments in standalone deploy mode provided by Spark. In addition, we used MySQL 5.7, MongoDB 3.2.9 and Neo4j 3.3.1 as DBMS. The results reported are the average values from six trials.

3.1 Experimental Setup

The goal of the experiments is to test the SQRE’s functionality of querying structured, semi-structured, and structure-less data from heterogeneous sources (e.g., MySQL, MongoDB, Neo4j, and XML files). However, the open data sets with meaningful integration scenarios are scarce. Therefore, we have designed two use cases for the purpose of our experiments.

In the first use case, we have created an artificial multi-store scenario using the Internet Movie Data Base (IMDB) provided by the Join Order Benchmark [11]. The original IMDB data set is in CSV files, containing 21 tables that occupy 3.6 GB in total. We have divided the 21 tables into three groups and imported them separately into MongoDB, Neo4j, and MySQL. MySQL holds the information regarding *movies* and Neo4j stores the *companies* related information. All the tables related to *type* are stored as collections in MongoDB. In this IMDB use case, we consider the original schema with 21 tables as the integrated schema, and the schemas in each DBMS as source schemas. The joins

Table 2. Data set statistics

Scenario	DS	Rec#	Ele#	Attr#
<i>Publications</i>	DBLP	1,466,351	19	15
	EPMC	4,399,876	36	32
	IntPubs		44	40
Drugs and proteins	DrugBank	8,257	427	290
	PSD	262,525	77	59
	IntDrugs		489	337
Startups and stocks	Startups	18,801	160	1230
	Stocks	6,756	72	69
	IntComps		226	193

Table 3. No. of returned records in Q1

MySQL <i>title</i>	Neo4j <i>movie_companies</i> ⋈ <i>company_name</i>	MongoDB <i>kind_type</i>	Final result
1.0 M	1.0 M ⋈ 234997	7	10106
1.3 M	1.3 M ⋈ 234997	7	15097
1.6 M	1.6 M ⋈ 234997	7	35845
1.9 M	1.9 M ⋈ 234997	7	127206
2.2 M	2.2 M ⋈ 234997	7	209830
2.5 M	2.5 M ⋈ 234997	7	275251

between data sets are performed using the existing foreign key constraints. We have compared the results returned by our system with the SQL query results with the same semantics, and verified the correctness and completeness of our results. We report the performance of SQRE for the IMDB use case in Sect. 3.2.

In the second use case, to examine the scalability of our query rewriting approach over complex data integration scenarios, we have designed three scenarios with two real-world data sets in each scenario. Table. 2 reports the statistics of the data sets (scenario, data source, #records, #elements, and #attributes in source schemas and integrated schema). Each scenario has two data sources sharing some attributes with the same semantics, but the two data sources may or may not describe the same entity.

In the *Publications* scenario, DBLP⁴ and Europe PubMed Central (EPMC)⁵ are two publication databases, both in JSON format. The next scenario uses Drugbank⁶ and the Protein Sequence Database (PSD)⁷, which share information regarding the protein name, citations, etc. Both data sets have been downloaded from the websites in XML format. The size of DrugBank is 587.2 MB and the size of PSD is 716.9 MB. In the *Startups and Stocks* scenario, the data of listed startup companies and stocks are downloaded in JSON format⁸. The size of datasets are 78.2 MB and 10.8 MB, respectively.

To prepare the input for the experiments for each scenario, we have conducted source schema extraction, schema matching, integrated schema generation, and nested mapping generation. Notably, different from the IMDB use case, for test queries in this use case we cannot obtain the ground truths directly. Thus, we went through the taunting task of manually checking the correctness and completeness of the results. The results of this use case are presented in Sect. 3.3.

⁴ <http://dblp.org/>.

⁵ <https://europepmc.org/>.

⁶ <https://www.drugbank.ca/>.

⁷ <http://aiweb.cs.washington.edu/research/projects/xmltk/xmldata/>.

⁸ <http://jsonstudio.com/resources/>.

3.2 Query Execution Performance

As we will point out in Sect. 4, existing systems mostly apply a different setting of multistore system from our work, or lack system details for us to conduct performance comparison. Therefore, for the purpose of comparison we have designed a baseline approach. It transforms data stored in MongoDB and Neo4j to CSV files, loads them into MySQL, and retrieves query results using MySQL. To examine our system both with and without joins in the local source data store, we have designed two queries, Q1 and Q2.

```

for $movie in collection("test", "IMDB")
for $type in collection("test", "IMDB")
let $movie_range := 2500000
where $movie.title.id lt $movie_range
  and $type.kind_type.id eq $movie.title.kind_id
group by $c:= $c.movie_companies.company_id,
  $Kind:= $type.kind_type.id
return { "Title": count($movie.title.title),
  "Kind": $type.kind_type.kind,
  "Company_Name": ( for $c in collection("test", "IMDB")
    let $c_range := 2500000
    where $c.movie_companies.id lt $c_range
      and $c.movie_companies.movie_id eq $movie.title.id
      and $c.movie_companies.company_id eq $c.company_name.id
    return $c.company_name.name) }
    
```

Fig. 6. Test query Q1

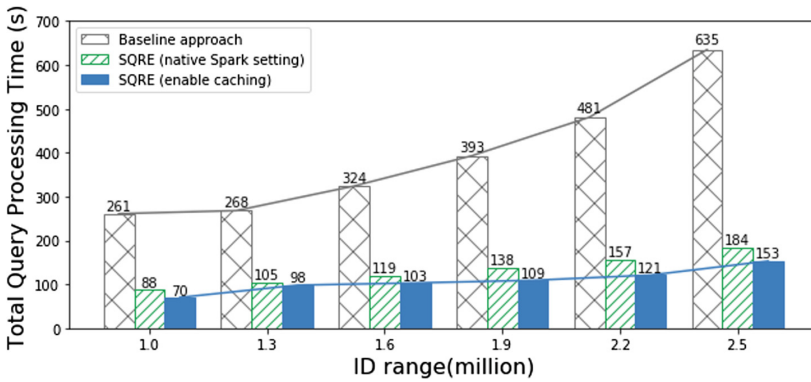


Fig. 7. Performance comparison with increasing returned data size in Q1

We first apply query Q1 in Fig. 6, which joins *title*, *kind_type*, *company_name* and *movie_companies*. The table *title* is stored in MySQL, while *kind_type* is stored as a collection in MongoDB and the other two tables are stored in Neo4j.

In order to see how the performance of SQRE varies with the increasing data size, we controlled the amount of records retrieved from MySQL and Neo4j by varying the ID range for *title* ($\$movie_range$) and *movie_companies* ($\$c_range$). We vary the ID range from 1 million to 2.5 million, which results also in an increasing dataset to be joined in Spark. Table 3 reports the size of the intermediate results of MySQL, Neo4j, MongoDB, and for the final results by Spark. Please note, since *company_name* and *movie_companies* are both stored in Neo4j, SQRE creates a subquery for Neo4j that joins these relations and only the result of the join is transferred to Spark. Thus, less records need to be loaded into Spark for the subsequent joins and aggregation functions.

```

for $m in collection("test","IMDB")
where $m.name.id eq
    $m.cast_info.person_id
    and $m.cast_info.id le 10000000
group by $ms:=$m.cast_info.movie_id,
    $genders:= $m.name.gender
return {
  "NumberOfActor":count($m.name.name),
  "Movie":$m.cast_info.movie_id,
  "Gender": $m.name.gender
}

```

Fig. 8. Test query Q2

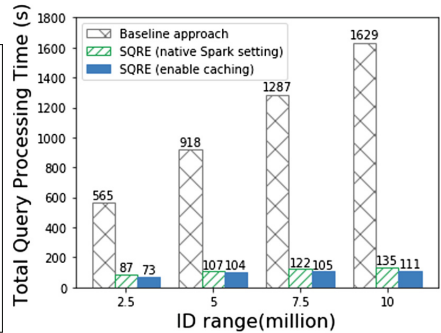


Fig. 9. Performance for Q2

Spark does not cache the intermediate results by default. Thus, we have implemented a method in SQRE to identify the intermediate views in Spark and cache them. Figure 7 depicts the total query processing time by the baseline approach and SQRE (with and without caching). SQRE outperforms the baseline approach with regard to the total query processing time. Moreover, by comparing the two settings in SQRE, we find that the performance of SQRE is further improved by caching. With an increasing dataset size, the query processing time of SQRE with caching shows a much smaller increasing slope than the baseline.

To measure the performance of SQRE with a larger data size, we designed a second query Q2 (up to 10 million records, cf. Fig. 8). Q2 retrieves two tables separately from MySQL and Neo4j (similar to the running example), on which the join and aggregation functions are performed later. Different from Q1, there is no join operation conducted in local data stores. We modified the ID range of table *cast_info* to obtain varying sizes of intermediate results and final results. Figure 9 shows the performance comparison with varying ID range in Q2. The results are similar as for Q1. With 10 million records retrieved from *cast_info*, the total query processing time of SQRE (enable caching) is only 111 s compared to 1629 s by the baseline approach, which is a $14.6\times$ improvement.

3.3 Query Rewriting Performance

We evaluated the performance of our query rewriting procedure including the query reformulation and translation from the given JSONiq query into multiple executable source subqueries and the final SparkSQL query. Due to space restrictions, we only report the results using the second use case. In order to isolate the query rewriting step, we have run SQRE and disabled the query execution, results integration, and transformation. We have conducted two experiments to study the impacts of query type and number of attributes, respectively. In the first experiment, for each pair of data sets in a data integration scenario, we tested four types of queries⁹ to see the impact of query complexity on query rewriting time: (1) *Q3*: return clause; (2) *Q4*: *Q3* + where clause; (3) *Q5*: *Q4* + group by clause; (4) *Q6*: *Q5* + nested query in the return clause. In this experiment, we fix the maximum number of attributes appearing in each query as 10. Each query is run five times and the mean value are shown in Fig. 10. In the majority cases of all scenarios, the query rewriting time slightly increases when queries get more complex. The query rewriting time for *Publications* scenarios is less than the other scenarios. The reason is that its two data sets, DBLP and EPMC, have less schema elements compared to the other data sets as reported in Table 2. The schema mappings for the *Publications* scenario are also simpler than the other two scenarios.

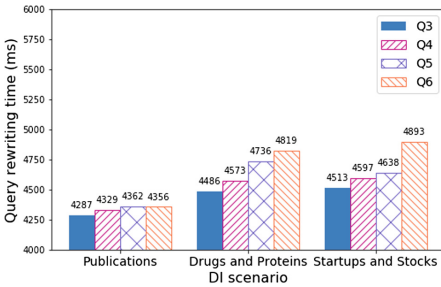


Fig. 10. Impact of different query types

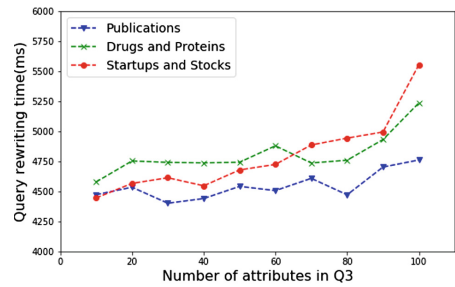


Fig. 11. Impact of attribute numbers

In the second experiment, we study the impact of query complexity regarding the total number of attributes in an input query. We use *Q3* with the number of attributes appearing in the queries varying from 10 to 100. If there are duplicated attributes appearing in the *return* clause, we rename them such that they are handled separately in query rewriting and returned as a single column in Spark. From the results shown in Fig. 11, we observe that the time increases sub-linear when the number of attributes in query rises. Even with a large amount of atomic elements (300) the query rewriting time is less than 5.6 s.

⁹ Queries and datasets are available at <https://bit.ly/2l9lXhc>.

The experiments show that our proposed query rewriting method scales well with increasing complexities of input queries and also with increasing dataset sizes. Thus, SQRE is a scalable query rewriting engine that could also handle well big data scenarios.

4 Related Work

Our approach is similar to recent multistore systems which also support diverse query languages and querying multiple data stores. We roughly divide the existing solutions into three categories. The first group of studies transforms the data in heterogeneous NoSQL stores into relational tables and uses an existing RDBMS to process the data. For example, Argo [2] studies enabling key-value and document-oriented systems in relational systems. Though such solutions offer an easy adaptation to existing relational systems, they come with a high cost of data transformation, especially with large data sets and are therefore not scalable for big data settings.

Another group of approaches focuses on multistore systems providing a SQL-like query language to query NoSQL systems. For instance, CloudMdsQL [9] supports graph, document and relational data stores. BigIntegrator [17] is a system that enables general queries over relational databases and cloud-based data stores (e.g., Google’s Bigtable). MuSQLE [5] performs optimization of distributed queries across multiple SQL engines. FORWARD [12] provides a powerful query language SQL++ for both structured and semi-structured data. Although these approaches perform more sophisticated optimizations than SQRE (e.g., semijoin rewriting), they do not consider schema mappings during the rewriting steps. We plan to include more complex optimization steps based on the intermediate logical representation of queries and schema mappings in the future.

The third category covers approaches which provide more efficient query processing techniques by applying a middle-ware to access the multiple NoSQL stores, such as Estocada [1] and Polybase [7], BigDAWG [3], and MISO [10]. In this paper, we focus on the setting of the multistore systems with heterogeneous data stores, e.g., relational, document-oriented (MongoDB), graph (Neo4j), or XML, and queries spanning multiple heterogeneous systems at the same time, which is not studied in most of the aforementioned works.

5 Conclusion

We presented SQRE, a spark-based query rewriting engine for unified query processing over heterogeneous data stores in data lake systems. We designed a general query parser that interprets queries in different formats into a unified logical representation. In particular, SQRE supports JSONiq queries and later translates them into executable queries for various relational and NoSQL systems. Moreover, our approach analyzes the predicates and rewrites them from integrated schema to source schemas using nested schema mappings. In SQRE,

we developed several translators that reformulate the original queries from internal representations into executable queries for SQL systems, MongoDB, Neo4j, and XML files. Further systems and query languages can be supported by implementing new translators and plugging them into our extensible architecture.

We have shown experimentally the usefulness and efficiency of our approach compared to a baseline approach through real world datasets. We have also shown that even with complex data integration scenarios, the query rewriting time of SQRE is scalable with increasing query complexity.

For future work, we will extend our system to support more query languages (e.g., Cypher, XQuery, etc.) and more NoSQL stores. To improve the system performance with large data sets, we want to explore the benefits of the distributed query processing. We are confident that the logical representation used in SQRE is well-defined basis for applying further query optimization techniques.

Acknowledgements. This work has been partially funded by the German Federal Ministry of Education and Research (BMBF) (project HUMIT, <http://humit.de/>, grant no. 01IS14007A), German Research Foundation (DFG) within the Cluster of Excellence “Integrative Production Technology for High Wage Countries” (EXC 128), and by the Joint Research (IGF) of the German Federal Ministry of Economic Affairs and Energy (BMWI, project charMant, <http://charmant-projekt.de/>, IGF promotion plan 18504N).




References

1. Bugiotti, F., et al.: Invisible glue: scalable self-tuning multi-stores. In: Proceedings of CIDR (2015)
2. Chasseur, C., Li, Y., Patel, J.M.: Enabling JSON document stores in relational systems. In: Proceedings of WebDB, pp. 1–6 (2013)
3. Duggan, J., et al.: The BigDAWG polystore system. SIGMOD Rec. **44**(2), 11–16 (2015)
4. Florescu, D., Fourny, G.: JSONiq: the history of a query language. IEEE Int. Comput. **17**(5), 86–90 (2013)
5. Giannakouris, V., Papailiou, N., Tsoumakos, D., Koziris, N.: MuSQL: distributed SQL query execution over multiple engine environments. In: Proceedings of Big Data, pp. 452–461 (2016)
6. Hai, R., Geisler, S., Quix, C.: Constance: an intelligent data lake system. In: Proceedings of SIGMOD, pp. 2097–2100 (2016)
7. DeWitt, D.J., et al.: Split query processing in polybase. In: Proceedings of SIGMOD, pp. 1255–1266. 22–27 June 2013
8. Jarke, M., Quix, C.: On warehouses, lakes, and spaces: the changing role of conceptual modeling for data integration. In: Cabot, J., Gómez, C., Pastor, O., Sancho, M., Teniente, E. (eds.) Conceptual Modeling Perspectives, pp. 231–245. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67271-7_16
9. Kolev, B., et al.: CloudMdsQL: querying heterogeneous cloud data stores with a common language. Distrib. Parallel Databases **34**(4), 463–503 (2016)
10. LeFevre, J., et al.: MISO: souping up big data query processing with a multistore system. In: Proceedings of SIGMOD, pp. 1591–1602 (2014)

11. Leis, V., et al.: How good are query optimizers, really? In: Proceedings of VLDB, pp. 204–215 (2015)
12. Ong, K.W., Papakonstantinou, Y., Vernoux, R.: The SQL++ unifying semi-structured query language, and an expressiveness benchmark of SQL-on-Hadoop, NoSQL and NewSQL databases. CoRR, abs/1405.3631 (2014)
13. Quix, C., Hai, R., Vatov, I.: Metadata extraction and management in data lakes with GEMMS. *Complex Syst. Inf. Model. Q.* **9**, 67–83 (2016)
14. Sharma, B., LaPlante, A.: Architecting data lakes. O’Reilly Media (2016). <https://resources.zaloni.com/ebooks/architecting-data-lakes>
15. Terrizzano, I., Schwarz, P.M., Roth, M., Colino, J.E.: Data wrangling: the challenging journey from the wild to the lake. In: Proceedings of CIDR (2015)
16. Yu, C., Popa, L.: Constraint-based XML query rewriting for data integration. In: Proceedings of SIGMOD, pp. 371–382 (2004)
17. Zhu, M., Risch, T.: Querying combined cloud-based and relational databases. In: 2011 International Conference Cloud and Service Computing (CSC) (2011)



RawVis: Visual Exploration over Raw Data

Nikos Bikakis^(✉) , Stavros Maroulis, George Papastefanatos ,
and Panos Vassiliadis 

University of Ioannina, Ioannina, Greece
bikakis.nikos@gmail.com

Abstract. Data exploration and visual analytics systems are of great importance in Open Science scenarios, where less tech-savvy researchers wish to access and visually explore big raw data files (e.g., json, csv) generated by scientific experiments using commodity hardware and without being overwhelmed in the tedious processes of data loading, indexing and query optimization. In this work, we present our work for enabling efficient query processing on raw data files for interactive visual exploration scenarios. We introduce a framework, named RawVis, built on top of a lightweight in-memory tile-based index, VALINOR, that is constructed on-the-fly given the first user query over a raw file and adapted based on the user interaction. We evaluate the performance of prototype implementation compared to three other alternatives and show that our method outperforms in terms of response time, disk accesses and memory consumption.

Keywords: In situ query · Big raw data · Adaptive processing
Visual analytics · Visualization · Indexing · User interaction
Exploratory data analysis

1 Introduction

In situ data exploration [1, 15–17] is a recent trend in raw data management, which aims at enabling on-the-fly scalable querying over large sets of volatile raw data, by avoiding the loading overhead of traditional DBMS techniques. A common scenario is that users wish to have a quick overview, explore and analyze the contents of a raw data file through a 2D visualization technique (e.g., scatter plot, map).

As an example, a scientist (e.g., astronomer) wishes to visually explore and analyze sky observations stored in raw data files (e.g., csv) using the Sloan Digital Sky Survey (SDSS) dataset (www.sdss.org), which describes hundreds of millions of sky objects (e.g., stars). First, the scientist selects the file and visualizes a part of it using a scatter plot with the sky coordinates (e.g., right ascension and declination). Then, she may focus on a sky region (e.g., defining

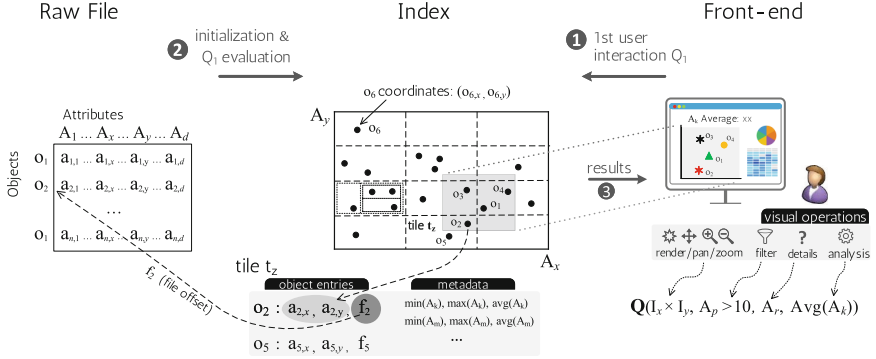


Fig. 1. RawVis framework overview

coordinates and area size), for which all contained sky objects are *rendered*; *move* (e.g., pan left) the visualized region in order to explore a nearby area; or *zoom-in/out* to explore a part of the region or a larger area, respectively. She may also click on a single or a set of sky objects and view *details*, such as *name* and *diameter*; *filter* out objects based on a specific characteristic, e.g., *diameter* larger than 50 km; or *analyze* data considering all the points in the visualized region, e.g., compute the average *age* of the visualized objects.

Most experimental and commercial visualization tools perform well for ad-hoc visualizations of small files (e.g., showing a trend-line or a bar chart) or over aggregated data (e.g., summaries of data points, into which user can zoom in), which *can fit in main memory*. For larger files, these tools usually require a preprocessing step for data to be *loaded* and either *indexed* in a traditional database, or *distributedly stored* and queried by *non-commodity hardware* (e.g., a Big Data tool).

On the other hand, in the in situ exploration scenarios, large data files which *do not fit in main memory*, must be efficiently handled *on-the-fly* using *commodity hardware* [1, 15–17]. In the in situ visual exploration scenarios, several challenges arise.

A first requirement is that no offline preprocessing is to be performed and any preprocessing, such as indexing or repartitioning must be performed *on-the-fly*, minimizing the user involvement and time overhead. Secondly, a non-expert user with limited programming or scripting skills must be supported to access and analyze data through visual ways, i.e., via an intuitive set of visual rather than data-access (e.g., querying) operations. Further, the response time of such visual operations must be significantly small (e.g., less than 1 s) in order to be acceptable by the user. Finally, the aforementioned operations have to be performed in machines with limited computational, memory and space resources, i.e., using commodity hardware.

In this work, we address the aforementioned challenges for enabling interactive 2D visual exploration scenarios of large raw data files using commodity hardware. We present the RawVis framework, which is built on top of a

lightweight main memory index, VALINOR (Visual AnaLysis Index On Raw data), constructed on-the-fly given the first user query and adapted based on the user interaction.

In our working scenario (Fig. 1), we assume that a user visually explores multidimensional objects stored in a raw file, using a 2D visualization technique. The user initially selects two attributes (A_x and A_y) as the X and Y axis of the visualization. In the first user interaction, the entire raw file is parsed and an “abstracted” version of the VALINOR is built on-the-fly, organizing the data objects into *tiles* based on their A_x and A_y values. Further, the index stores auxiliary metadata in each tile regarding its contents (e.g., average attribute values), in order to reduce computation cost and access to the raw file. Throughout the exploration, visual user operations (e.g., pan, zoom, filter) are expressed as queries evaluated over VALINOR. Following the user interaction, VALINOR incrementally reorganizes the objects’ grouping, constructs tile hierarchies, and recomputes and enriches metadata.

The main contributions of this work are: (1) we formulate visual user interactions as data-access operations; (2) we design an index in the context of in situ visual exploration over large data; (3) we describe the query processing techniques over this index; (3) we conduct an experimental evaluation using real and synthetic datasets, as well as several systems and structures; i.e., MySQL, PostgresRaw, and R-tree, which shows that our technique outperforms competitors both in execution time and memory consumption.

2 Preliminaries

Data File and Objects. We assume a *raw data file* \mathcal{F} containing a set of d -dimensional objects \mathcal{O} . Each dimension j corresponds to an *attribute* $A_j \in \mathcal{A}$, where each attribute may be numeric or textual. Each object o_i contains a list of d attributes $o_i = (a_{i,1}, a_{i,2}, \dots, a_{i,d})$, and it is associated with an *offset* f_i in \mathcal{F} pointing to the “position” of its first attribute, i.e., $a_{i,1}$. Note that, object entries can be either fixed or variable-length, in the latter case they are separated by a special-character; e.g., CR for a text file, that precedes the offsets.

Visual Exploration Scenario. Given a set of d -dimensional objects, the user arbitrarily selects two attributes $A_x, A_y \in \mathcal{A}$, whose values are numeric and can be mapped to the X and Y axis of a 2D visualization layout. A_x and A_y attributes are denoted as *axis attributes*, while the rest as *non-axis*.

The user follows a sequence or combination of the following *visual operations* to interact with the data: (1) *render*: visualizes the objects included in a specified 2D area, denoted as *visualized area*. (2) *move*: changes the visualized area (i.e., pan). (3) *zoomin/out*: specifies a new visualized area that is within (resp. covers) the current visualized area. (4) *filter*: excludes objects from the visualized area, based on conditions over non-axis attributes. (5) *details*: presents values of non-axis attributes. (6) *analyze*: analyzes data from the objects included in the visualized area.

Note that, as previously illustrated, multiple visual operations can be performed in a single user interaction; e.g., zoom in a region while filtering the presented objects.

Exploratory Query. Considering the aforementioned visual operations, we define the corresponding data-access operators. In what follows, we formulate the notion of an exploratory query. Given a set of d -dimensional objects \mathcal{O} and the axis attributes A_x and A_y , an *exploratory query* Q over \mathcal{O} is defined by the tuple $\langle S, F, D, N \rangle$, where:

- The *Select part* S defines a 2D range query (i.e., window query) specified by two intervals $S.I_x$ and $S.I_y$ over the axis attributes A_x and A_y , respectively. This part selects the objects $\mathcal{O}_S \subseteq \mathcal{O}$ for each of which both of their axis attributes have values within the respective intervals, defined by the window query. The select part is *mandatory*, while the rest parts are *optional*.
- The *Filter part* F defines conditions over the non-axis attributes. As \mathcal{A}_F we denote the set of attributes involved in F . In our current implementation, the conditions in F are expressed using a single attribute, unary and binary arithmetic operations, and constants. The filter part is applied over the objects \mathcal{O}_S , selecting the objects $\mathcal{O}_Q \subseteq \mathcal{O}_S$ that satisfy F . If the filter part is not defined (i.e., $F = \emptyset$), then $\mathcal{O}_Q = \mathcal{O}_S$.
- The *Details part* D defines a set \mathcal{A}_D of non-axis attributes. The query returns the values of these attributes for each object in \mathcal{O}_Q .
- The *Analysis part* N defines aggregate, analytic, or user-defined functions over numeric attributes of the objects \mathcal{O}_Q . As \mathcal{A}_N we denote the attributes that are used in these functions. In our current implementation, we consider a single attribute and the following aggregate functions: count, sum, average, min, and max. The analysis part returns the values \mathcal{V}_N from the evaluation of the specified functions.

The semantics of query execution involves the evaluation of the four parts in the following order: (1) *Select* part; (2) *Filter* part; (3) *Details*; (4) *Analysis* part.

Query Result. An exploratory query Q returns the axis values for the objects \mathcal{O}_Q along with their values \mathcal{V}_D of the attributes specified in D , denoted as \mathcal{O}_Q^D ; and the numeric values \mathcal{V}_N resulted from the analytic part. Formally, the result is consisted by: (1) a set of tuples $\mathcal{O}_Q^D = \langle \alpha_{i,x}, \alpha_{i,y}, \alpha_{i,A_{D_1}}, \dots, \alpha_{i,A_{D_d}} \rangle, \forall o_i \in \mathcal{O}_Q, \forall d \in \{1, \dots, |\mathcal{A}_D|\}$ with $A_{D_d} \in \mathcal{A}_D$; and (2) a list of numeric values \mathcal{V}_N .

Example (*From Visual operations to Exploratory query*). Each visual operation can be expressed as an exploratory query. Specifically, the *render* operation is implemented using only the *Select* part of the query, setting the intervals equal to the values of the visualized area. Assuming our running example, in which $A_x = \textit{declination}$ and $A_y = \textit{right ascension}$. A *render* operation that visualizes

the rectangle sky area from 100° to 110° , and from 20° to 25° , is executed by defining $S.I_x = [100^\circ, 110^\circ]$ and $S.I_y = [20^\circ, 25^\circ]$.

The *move*, *zoom in/out* operations are also implemented by defining a *Select* part, having as parameters the coordinates of the neighboring, contained/containing visualized regions, respectively. For example, a *zoom-out* operation over the previously presented area is executed as $S.I_x = [97.5^\circ, 112.5^\circ]$ and $S.I_y = [18.75^\circ, 26.25^\circ]$. Further, the *filter* operation is implemented by including a *Filter* part. In our example $F = \text{“diameter} > 50 \text{ km”}$. Finally, the *details* and *analyze* operations correspond to the *Details* and *Analysis* parts. For example the details and analyze operations described in our example correspond to $D = \{name, diameter\}$ and $N = \text{“avg(age)”}$, respectively.

3 The VALINOR Index

The VALINOR is a lightweight *tile-based multilevel* index, which is stored in memory, organizing the data objects of a raw file, into *tiles*. The index is constructed on-the-fly given the first user query and incrementally adjusts its structure to the user visual interactions. In the construction, each tile is defined on a fixed range of values of the A_x and A_y axis attributes, by dividing the euclidean space (defined by the A_x and A_y domains) into initial tiles. Then, user operations split these tiles subsequent into more fine-grained ones, thus forming a hierarchy of tiles. In each level of the hierarchy, all tiles are disjoint (i.e., non-overlapping) and can belong to only one parent tile. Next we formalize the main concepts of the proposed index.

Object Entry. For an object o_i its *object entry* e_i is defined as $\langle a_{i,x}, a_{i,y}, f_i \rangle$, where $a_{i,x}, a_{i,y}$ are the values of the axis attributes, f_i the offset of o_i in the raw file.

Tile. A *tile* t is a part of the Euclidean space defined by two intervals $t.I_x$ and $t.I_y$. Each t is associated with a *set of object entries* $t.\mathcal{E}$, if it is a leaf tile, or a set of *child tiles* $t.C$, if it is a non-leaf tile. The set $t.\mathcal{E}$ is defined as a set of object entities, such that for each $e_i \in t.\mathcal{E}$ its attribute values $a_{i,x}$ and $a_{i,y}$ fall within the intervals of the tile t , $t.I_x$ and $t.I_y$ respectively. Further, t is associated with a set $t.\mathcal{M}$ of *metadata* related with the $t.\mathcal{E}$ objects contained in the tile, e.g., aggregated values over attributes. As $t.\mathcal{M}_A$ we denote the attributes for which metadata has been computed for the tile t .

VALINOR Index. Given a raw data file \mathcal{F} and two axis attributes A_x, A_y , the index organizes the objects into non-overlapping rectangle tiles based on its A_x, A_y values. Specifically, the VALINOR *index* \mathcal{I} is defined by a tuple $\langle \mathcal{T}, IP, AP, MH \rangle$, where \mathcal{T} is the set of *tiles* contained in the index; *IP* is the *initialization policy* defining the initial tile size; *AP* is the *adaptation policy* defining a criterion (e.g., the relation of a tile’s size w.r.t. the query’s window size), and a

method for splitting tiles and reorganizing object entries following user’s interaction; and MH is the *metadata handler* defining and computing the metadata stored in each tile.

VALINOR Initialization.

In our approach we do not consider any preprocessing for the index construction, but rather the index is constructed on-the-fly upon the first time the user requests to visualize a part of the raw file. The file is scanned once, for creating the initial VALINOR structure and computing the results of the

first query. The initial version of VALINOR corresponds to a flat tile structure that does neither exhibit any hierarchy nor contain any metadata to the tiles. This phase, referred to as *initialization phase*, aims at minimizing the response time of the first user action, by avoiding computations which may not be used through exploration.

Algorithm 1 describes the initialization phase. The input *initialization policy* IP determines the initial size of the tiles $x_0 \times y_0$ (line 1). Algorithm 1 first scans the raw file \mathcal{F} once (loop in line 3), reads the values of $a_{i,x}$, $a_{i,y}$ f_i of each object o_i (line 4 & 5), and appends a new object e_i to the entries $t.\mathcal{E}$ of the tile t (line 6). It, finally, evaluates whether this entry should be included in the results of the initial query.

The initialization policy IP is a parameter that defines the initial tile size. It can be given either explicitly by the user (e.g., in a map the user defines a default scale of coordinates for the initial visualization), be computed based on data characteristics (e.g., the ranges of the A_x , A_y attributes), or adjusted to the visualization setting considering certain characteristics (e.g., screen size/resolution) [4, 13].

4 Query Processing and Index Incremental Adaptation

This section describes the evaluation of exploratory queries over the proposed index.

Query Processing Workflow. Algorithm 2 outlines the workflow of the query evaluation. Given a query Q , we first look up the index \mathcal{I} and determine the tiles \mathcal{T}_Q that overlap with the query. In addition, we examine their objects and select the ones \mathcal{O}_S (line 1) that are contained in the query window.

Algorithm 1. Initialization (\mathcal{F} , A_x , A_y , Q)

Input: \mathcal{F} : raw data file; A_x, A_y : X and Y axis attributes;
Parameters: IP: initialization policy
Output: \mathcal{T} : initialized index tiles;
 $\mathcal{O}_Q^D, \mathcal{V}_N$: first query results

```

1  $\mathcal{T} \leftarrow \emptyset$  //set of tiles
2  $x_0, y_0 \leftarrow \text{IP.getInitialTileSize}()$  //initial tile size
3 foreach  $o_i \in \mathcal{F}$  do //read objects from raw file
4   read  $a_{i,x}, a_{i,y}$  from  $\mathcal{F}$ 
5   set  $f_i \leftarrow$  offset of  $a_{i,1}$  in  $\mathcal{F}$ 
6   append  $\langle a_{i,x}, a_{i,y}, f_i \rangle$  to tile  $t.\mathcal{E}$ , where  $t \in \mathcal{T}$ 
   determined from  $a_{i,x}, a_{i,y}$  and  $x_0, y_0$ 
7   compute first query result ( $\mathcal{O}_Q^D, \mathcal{V}_N$ ) through file parsing
8 return  $\mathcal{T}, \mathcal{O}_Q^D, \mathcal{V}_N$ 

```

Considering \mathcal{T}_Q , we determine the tiles $\mathcal{T}_{\mathcal{F}}$ for which we have to access the raw file in order to answer the query (line 3). In this step, we also split the tiles $t \in \mathcal{T}_{\mathcal{F}}$ based on the adaptation policy AP , creating a new set of tiles W , as explained later in the section. Next, we access the file at each offset of the objects in $\mathcal{T}_{\mathcal{F}}$ tiles and retrieve in memory the attributes defined in the *details*, *filter*, and *analysis* parts (line 7).

We use these values for the metadata handler MH to compute and store metadata in each tile (line 8). Finally, we evaluate the filter and analysis parts on the retrieved objects (lines 9 & 10). Each different operation of Algorithm 2 is described below.

Algorithm 2. Query Processing ($\mathcal{I}, Q, \mathcal{F}$)

Input: \mathcal{I} : index; Q : query; \mathcal{F} : raw data file
Variables: \mathcal{O}_S : objects selected from select part;
 \mathcal{T}_Q : tiles that overlapped with the select part;
 $\mathcal{T}_{\mathcal{F}}$: tiles for which file access is required
Parameters: AP : adaptation policy; MH : metadata handler
Output: \mathcal{O}_Q^D : objects of the result along with the detail values;
 \mathcal{V}_N : analysis values

```

1  $\mathcal{O}_S, \mathcal{T}_Q \leftarrow \text{evaluateSelectPart}(\mathcal{I}, S)$ 
2 foreach  $t \in \mathcal{T}_Q$  do
3   if  $\text{fileAccessRequired}(t, Q)$  then
4      $W \leftarrow AP.\text{splitTile}(t)$ 
5     add  $W$  into  $\mathcal{T}_{\mathcal{F}}$ 
6 if  $\mathcal{T}_{\mathcal{F}} \neq \emptyset$  then
7    $\mathcal{V}_{AF}, \mathcal{V}_{AD}, \mathcal{V}_{AN} \leftarrow \text{readFile}(\mathcal{T}_{\mathcal{F}}, \mathcal{F}, \mathcal{O}_S)$ 
8    $MH.\text{updateMetadata}(\mathcal{T}_{\mathcal{F}}, Q, \mathcal{V}_{AF}, \mathcal{V}_{AN})$ 
9    $\mathcal{O}_Q^D \leftarrow \text{evaluateFilterPart}(\mathcal{O}_S, \mathcal{F}, \mathcal{V}_{AF}, \mathcal{V}_{AD})$ 
10  $\mathcal{V}_N \leftarrow \text{evaluateAnalysisPart}(\mathcal{O}_Q, \mathcal{V}_{AN})$ 
11 return  $\mathcal{O}_Q^D, \mathcal{V}_N$ 

```

Evaluate Select Part. In order to evaluate the Select part over the index (Algorithm 2, line 1), we have to identify \mathcal{O}_S by accessing the leaf tiles \mathcal{T}_Q which overlap with the window query specified in Q . Since window queries can be evaluated directly from the index, the Select part is computed without performing any I/Os.

Determine File Read. Procedure 1 determines whether and for which objects, file reads are required. The check is performed for each tile t independently and distinguishes between the type of operation is requested by the query. If details are requested, we always have to access the file (line 1); otherwise for each t we check if the Analysis and Filter parts can be evaluated using the metadata that has been already computed for t (line 5), by a previous query.

Procedure 1. fileAccessRequired(t, Q)

Input: t : tile; Q : query;
Output: true / false: file access is required

```

1 if  $D \neq \emptyset$  then //detail part is included
2   return true //access file for the  $\mathcal{O}_Q$  objects in  $t$ 
3 else if  $F = \emptyset$  and  $N = \emptyset$  then //no filter & analysis parts
4   return false
5 else if  $\mathcal{A}_N \subseteq t.\mathcal{M}_A$  and  $F$  can be evaluated from
    $t.\mathcal{M}_A$  then //filter and/or analysis part is included
6   return false
7 else
8   return true //access file for the  $\mathcal{O}_Q$  objects in  $t$ 

```

Incremental Computation of Metadata and Query Processing. The metadata is computed incrementally and help improving the performance evaluation of the filter expressions and the functions defined in the analysis part.

Procedure 2. splitTile(t)

Input: t : tile
Parameters: AP: adaptation policy
Output: \mathcal{T}_a : tiles resulted from adaptation

```

1 if AP.checkCondition( $t$ ) =
   reconstruction required then
2   |  $\mathcal{T}_a \leftarrow$  AP.reconstruct( $t$ )
3 else
4   |  $\mathcal{T}_a \leftarrow t$ 
5 return  $\mathcal{T}_a$ ;
```

Procedure 3. readFile($\mathcal{T}_F, \mathcal{F}, \mathcal{O}_S$)

Input: \mathcal{T}_F : tiles for which file access is required;
 \mathcal{F} : raw data file \mathcal{O}_S : data selected from the select part
Output: $\mathcal{V}_{A_F}, \mathcal{V}_{A_D}, \mathcal{V}_{A_N}$, attributes values required for the filter, details & analysis part

```

1 forall the  $o_i$  included in tiles  $\mathcal{T}_F$ , with  $o_i \in \mathcal{O}_S$  do
2   | access  $\mathcal{F}$  at file offset  $f_i$ 
3   |  $\mathcal{V}_{A_F}, \mathcal{V}_{A_D}, \mathcal{V}_{A_N} \leftarrow$  read the  $o_i$  attributes values
   | that are required for the F, D and N parts
4   | insert:  $\mathcal{V}_{A_F}$  into  $\mathcal{V}_{A_F}$ ;  $\mathcal{V}_{A_D}$  into  $\mathcal{V}_{A_D}$ ;  $\mathcal{V}_{A_N}$  into  $\mathcal{V}_{A_N}$ 
5 return  $\mathcal{V}_{A_F}, \mathcal{V}_{A_D}, \mathcal{V}_{A_N}$ 
```

Incremental computation implies that metadata for a tile is not fully computed during the (initial) index construction, but rather, during user exploration, i.e., the first time a query access a tile.

For example, the Analysis part of a user query requires the aggregate (min, sum, etc.) value of a non-axis attribute that was already computed for the tiles overlapping with the query, from a previous user visit. In this case, we do not need to read the non-axis attribute from the file, as we can aggregate the precomputed values for computing the output value of the query. Further, assume that we have a Filter part, e.g., $F = \text{“diameter} > 50 \text{ km”}$, evaluated over the objects of a tile t , on which we have stored as metadata the maximum value of the *diameter* if its containing objects, e.g., 60 km. Similarly, we can easily answer the query through a single lookup at the index and avoid processing the objects in t . Currently, we consider aggregate values over attributes of the $t.\mathcal{E}$ objects as metadata – i.e., count, sum, average, min, max.

The metadata handler MH, using the values retrieved from file, for each tile (Algorithm 2, line 8): (1) determines for which attributes to compute metadata; (2) computes metadata; and (3) updates metadata in the tiled accessed by the query. The attributes for which metadata is computed are: the attributes $t.\mathcal{M}_A$ for which metadata has been previously computed in t , as well as the attributes included in the query filter \mathcal{A}_F and analysis part \mathcal{A}_N .

Incremental Index Adaptation. Procedure 2 reorganizes objects in the index by splitting tiles into smaller ones, based on the adaptation policy AP. It first checks whether a tile t must be split (*checkCondition*, line 1) and, if so, places all objects in t into the new tiles T that are resulted from splitting (line 2). Note that, a tile’s splitting (if required) is performed, each time (i.e., *incrementally*) a query accesses the particular tile (i.e., *adaptively*). The incremental adaptation attempts to maximize the number of tiles which are fully-contained in a query window. For a fully-contained tile t , there is no need to: (1) access the file if the required metadata have been computed for t ; and (2) examine the objects in t in order to find the objects that are included in the window. Thus, fully-contained tiles reduce both computation and I/O cost.

In our current implementation, AP defines a numeric *threshold* $h > 0$, defining the relation between the tile and the window size. The *checkCondition* method

(*line 1*) examines if the size of the t is more than $1/h$ times larger than the window size. Then, using the *reconstruct* method (*line 2*), t is *split* into more tiles constructing a tile hierarchy. The splitting is performed following the method used in either Quadtree or k-d tree.

Read File. We use the file offset stored in object entries, in order to access the file (Procedure 3). For each object we read the attributes values required for the filter, details & analysis part. A crucial issue in our index is to improve the execution time of the queries when file access is required. Exploiting the way that VALINOR constructs and stores the object entries, we are able to scan the raw file in a sequential manner. The sequential file scan increases the number of I/Os over continuous disk blocks and improves the utilization of the look-ahead disk cache.

During the initialization phase, the object entries are appended into tiles as the file is parsed (Algorithm 1, *line 5*). As a result, the object entries in each tile are sorted based on object’s file offset. In the query evaluation, we identify the tiles $\mathcal{T}_{\mathcal{F}}$ for which we have to read the file (Algorithm 2, *line 3*). Considering the lists of objects entries in $\mathcal{T}_{\mathcal{F}}$, we read the objects from lists following a k -way merge, i.e., all objects are sorted on their offset before reading the file. This way, objects values are read by accessing the file in sequential order. Note that, in our experiments, the sequential access results in about 8-times faster I/O operations compared to accessing the file by reading objects on a tile basis (i.e., read the objects of tile w , then read the objects of tile v , etc.).

Evaluate Filter, Details and Analysis Parts. In the general case, the *Filter part* requires access to the file to retrieve the values of the attributes included in the filter conditions. However, there are cases where precomputed values in metadata (e.g., min, max) can be exploited to avoid files access. On the other hand, the *Details part* always requires access to the raw file (Procedure 3), since our index does not keep in memory attribute values other than the two axis attributes. Particularly, for each object o_i in, using the file offset pointers, we retrieve the values \mathcal{V}_{D_i} of the attributes included in the details part. Finally, the *Analysis part* is first evaluated on existing metadata stored in the index; otherwise it requires access to the values \mathcal{V}_{A_N} read from the file.

5 Experimental Analysis

5.1 Experimental Setup

Datasets. We have used a *real dataset*, the “Yahoo! Flickr” (YAHOO), which is a csv file, containing information regarding public Flickr photos/videos¹. YAHOO contains 100M objects and each object refers to a photo/video described by 25

¹ Available at: <https://research.yahoo.com>.

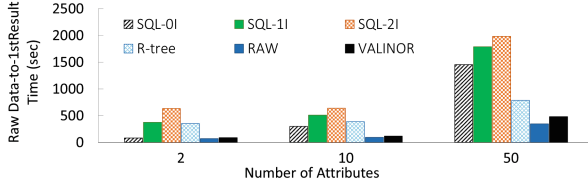


Fig. 2. From-Raw Data-to-1stResult Time (SYNTH dataset)

attributes (e.g., longitude, latitude, accuracy, tags). We consider a map-based visualization, with the axis attributes being the longitude and latitude of the location where a photo was taken. From this dataset, we select the photos/videos posted in USA region, resulting to 13M *data objects* and a csv file of 7 GB. Regarding the *synthetic datasets* (SYNTH), we have generated csv files of 100M *data objects*, having 2, 10, and 50 *attributes* (2, 11, and 51 GB, respectively), with 10 being the default value. Each attribute value is a real number in the range (0, 1000) and follows a uniform distribution.

Competitors. We have compared with: (1) A traditional DBMS (MySQL 5.5.58), where the user has to load all data in advance in order to execute queries; three indexing settings are considered: (a) no indexing (SQL-0I); (b) one composite B-tree on the two axis attributes (SQL-1I); and (c) two single B-trees, one for each of the two axis attributes (SQL-2I). MySQL also supports SQL querying over external files (see CSV Storage Engine in Sect. 6); however, due to low performance [1], we do not consider it as a competitor in our evaluation. (2) PostgresRaw (RAW)² (build on top of Postgres 9.0.0) [1], which is a generic platform for in situ querying over raw data (see Sect. 6). (3) A main memory Java implementation of the R*-tree³. We have tested various configurations for the index fan-out, ranging from 4 to 128; as the difference in the performance is marginal, we only report on the best one, i.e., 16.

Implementation. We have implemented VALINOR⁴ in Java and the experiments were conducted on an 2.67 GHz Intel Xeon E5640 with 64 GB of RAM. We applied memory constraints (max Java heap size) in order to measure the performance of our approach and our competitors in a commodity hardware setting. For large datasets, the available version of RAW, required a significant amount of memory (in some cases more than 32 GB); the same held for the in-memory R-Tree implementation (more than 16 GB in most cases). In contrast, VALINOR performed well for heap size less than 10 GB for the larger dataset of 100M objects, 50 attributes (51 GB).

² <https://github.com/HBPMedical/PostgresRAW>.

³ <https://github.com/davidmoten/rtree>.

⁴ The source code is available at <https://github.com/Ploigia/RawVis>.

Evaluation Scenario and Metrics. We study the following visual exploration scenarios: (1) first, the user requests to view a region of the data from the raw file. For this action, referred to as “From-Raw Data-to-1stResult”, we measure the time for creating the index and fetching the query results. (2) Next, the user explores neighboring areas, using render, move and zoom operations, denoted as “Basic Visual Operations”, for which we measure the query response time. (3) Finally, the user explores neighboring areas of the dataset, requesting also aggregate values for the included objects. For that, we examine the efficiency of our adaptive method, over a sequence of overlapping window queries.

In our experiments, we measured *time*, *memory consumption* and *file accesses* varying the following parameters: *cardinality* (number of objects), *dimensionality* (number of attributes), and *query selectivity* (i.e., objects included in the examined area).

5.2 Results

From-Raw Data-to-1stResult Time. In this experiment, we measured the time required to answer the first query. This corresponds to loading and indexing the data for MySQL, and to constructing the positional map for RAW. For the VALINOR and R-tree cases the in-memory indexes must be built. For the R-tree construction, bulk-loading was used. In VALINOR we used 100×100 tiles for the initialization of the index (this number of tiles is used in all the experiments). Note that, we also examined different number of tiles; however, the effect on the performance was negligible and we do not report on these results. Figure 2 presents the results varying the dimensionality of the objects. VALINOR outperforms the MySQL and R-tree methods, with the difference getting more significant as the dimensionality increases. As expected, VALINOR exhibits a lower initialization time than R*-tree; the latter must determine multilayer MBRs and assign objects to leaf nodes as opposed to our approach which is initialized with fixed tile sizes exhibiting no hierarchy. In this experiment, VALINOR is outperformed by RAW, due to its non-optimized CSV parsing and slower I/O Java operations, as opposed to the efficiency provided by RAW’s programming language (i.e., C). This is something we plan to address in the future.

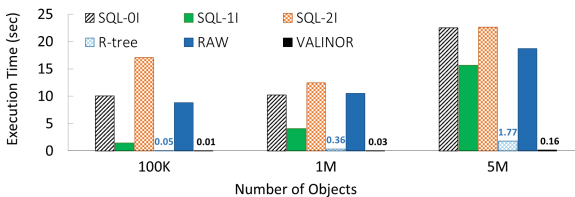


Fig. 3. Execution time for basic visual operations (SYNTH dataset)

Performance of Basic Visual Operations. In this experiment we study the performance of the *render*, *move* and *zoom in/out* visual operations. Recall that, these operations are expressed as queries over the axis attributes (i.e., windows queries). We use the SYNTH dataset to evaluate these operations over regions that contain different numbers of objects (100K, 1M, 5M). Figure 3 presents the query execution time. As expected, execution time increases for higher values of selectivity. VALINOR significantly outperforms all methods in all cases.

Regarding RAW, its positional map is used to minimize file parsing and tokenizing costs, which can not be exploited to reduce the number of objects examined in range queries. R-tree is the best competing method. However, its performance is significantly affected by the number of objects. Overall, VALINOR is around 5–12× faster than R-tree, and more than 1 magnitude faster than the other methods.

Index Adaptation. We define a sequence of neighboring and overlapping queries in order to study the adaptation of the index, and we measure the execution time for each query. To assess the effect of VALINOR’s adaptation policy, we compare its performance with that of a “static” VALINOR version (denoted as VALINOR-S), for which tiles are not split as a result of user queries, and we measure the number of objects read from the file for each version. As adaptation policy we consider standard Quadtree splitting of tiles and we only present MySQL-II, which has exhibited the best performance.

First, we use the *real* YAHOO dataset (Fig. 4). We constructed a sequence of ten queries (Q1-Q10), each one defined over an area of 10 km × 10 km size (i.e., window size), requesting the average value of one of the non-axis attributes. Every query is *shifted* by 10% of the window size (i.e., 1 km) in relation to its previous one, where the shift direction (N, E, S, W) is randomly selected, with the first query Q1 posed in central LA. Note that, we were not able to run RAW in this dataset, due to the types of the attributes included in YAHOO (i.e., textual). For the *synthetic* dataset (Fig. 5), we use a default window size with approximately 100K objects, and uniform data distribution, shifting each query by 10% of the window size.

Comparing VALINOR with its non-adaptive version, we observe that in both datasets (Figs. 4 and 5) the adaptive method exhibits better performance in time and file reads. The number of objects read from file is defined in the right axis and depicted with slashed lines. This improvement, as a result of the tile splittings and the computed aggregate values, is obvious even after the first query Q1, with the difference getting more significant after the query Q3. Variations in these improvements are due to the different number of objects contained in each window query, as well as to its position of the query w.r.t. to its previous one. Overall, for the real dataset, the adaptive method requires up to 6× less file reads (slashed lines) and up to 5× less time; 25× and 17× for the synthetic one, respectively.

Compared to the other methods, VALINOR outperforms all methods with the exception of the first 3 queries in the YAHOO dataset (Fig. 4) where SQL

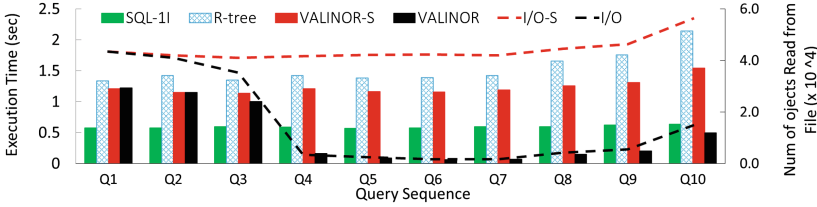


Fig. 4. Execution Time & File Accesses for each Query (YAHOO dataset)

performs better. Regarding the RAW method (Fig. 5), we observe that it requires approximately the same time for every query, since it does not adapt to the workload and the positional map cannot be exploited to answer the aggregate queries. Considering all the workload (Q1-Q10), in YAHOO, VALINOR requires 4s to execute all queries, VALINOR-S 12s, R-tree 15s, and SQL-II 6s; in SYNTH, 74, 82, 105 and 145s, respectively.

Memory Consumption. We measured the memory used to build VALINOR and R-tree varying the number of objects in the SYNTH dataset (Fig. 6). Note that, the memory consumption in VALINOR and R-tree is not affected by the objects’ dimensionality, since in each case, only the two axis attributes are indexed. We did not consider RAW and MySQL settings since they exhibit different memory requirements due to their tight-coupling with the DBMS. We can observe that VALINOR requires significantly less memory than R-tree, with R-tree requiring $2\times$ more memory for 100M objects.

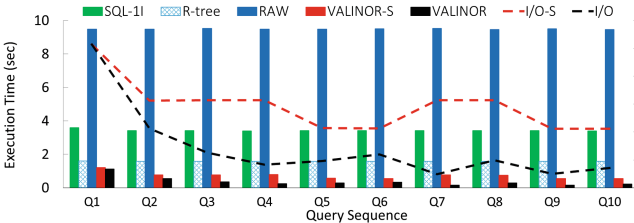


Fig. 5. Execution Time & File Accesses for each Query (SYNTH dataset)

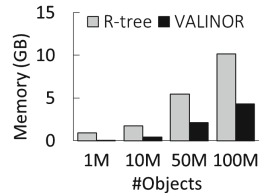


Fig. 6. Memory Consumption (SYNTH dataset)

6 Related Work

In situ Query Processing. Data loading and indexing usually take a large part of the overall time-to-analysis for both traditional RDBMS and Big Data systems [11]. In situ query processing aims at avoiding data loading in a DBMS

by accessing and operating directly over raw data files. NoDB [1] is one of the first efforts for in situ query processing. NoDB incrementally builds on-the-fly auxiliary indexing structures called “positional maps” which store the file positions of data attributes in order to reduce parsing and tokenization costs during query evaluation, as well as it stores previously accessed data into cache. The authors have also developed PostgresRaw, which is an implementation of NoDB over PostgreSQL. DiNoDB [17] is a distributed version of NoDB. In the same direction, RAW [15] extends the positional maps to index and query file formats other than CSV. Recently, Slalom [16] exploits the positional maps and integrates partitioning techniques that take into account user access patterns.

Raw data access methods have been also employed for the analysis of scientific data, usually stored in array-based files. In this context, Data Vaults [12] and SDS/Q [5] rely on DBMS technologies to perform analysis over scientific array-based file formats. Further, SCANRAW [6] considers parallel techniques to speed up CPU intensive processing tasks associated with raw data accesses.

Recently, several well-known DBMS support SQL querying over raw csv files. Particularly, MySQL provides the CSV Storage Engine, Oracle offers the External Tables, and Postgres has the Foreign Data. However, these tools do not focus on user interaction, parsing the entire file for each posed query, and resulting in significantly low query performance [1] for interactive scenarios.

All the aforementioned works study the generic in situ querying problem without focusing on the specific needs for raw data visualization and exploration. Instead, our work is the first effort trying to address these aspects, considering the in situ processing of a specific query class, that enables user operations in 2D visual exploration scenarios; e.g., pan, zoom, details. The goal of our solution is to optimize these operations, such that visual interaction with raw data is performed efficiently on very large input files using commodity hardware.

Indexes for Visual Exploration. VisTrees [9] and HETree [4] are tree-based main-memory indexes that address visual exploration use cases; i.e., they offer exploration-oriented features such as incremental index construction and adaptation. Compared to our work, both indexes focus on one-dimensional visualization techniques (e.g., histograms), and they do not consider disk storage; i.e., data is stored in-memory.

Hashedcubes [8] is a main-memory data structure supporting a wide range of interactive visualizations, such as heatmaps, time series, plots. It is based on multiple hierarchical multidimensional (spatial, categorical, temporal) indexes, which are constructed during the loading phase. The construction requires multiple sortings on the input values, which may result in increased amount of time for large datasets. In comparison with our approach, Hashedcubes requires that all data resides in memory, and thus it does not address the need of reducing the overall time-to-visualization (both loading and query processing time) over raw data files and it does not feature any adaptive technique based on the user interaction. Further, graphVizdb [3] is a graph-based visualization tool, which employs a 2D spatial index (e.g., R-tree) and maps user interactions into window

2D queries. Compared to our work, graphVizdb requires a loading phase where data is first stored and indexed in a relational database system. In addition, it targets only graph-based visualization and interaction, whereas our approach offers interaction in 2D layouts, such as maps or scatter diagrams.

In different contexts, tile-based structures are used in visual exploration scenarios. Semantic Widows [14] considers the problem of finding rectangular regions (i.e., tiles) with specific aggregate properties in an interactive data exploration scenario. This work uses several techniques (e.g., sampling, adaptive prefetching, data placement) in order to offer interactive online performance. ForeCache [2] considers a client-server architecture in which the user visually explores data from a DBMS. The approach proposes a middle layer which prefetches tiles of data based on user interaction. Prefetching is performed based on strategies that predict next user’s movements. Our work considers different problems compared to the aforementioned approaches. However, some of their methods can be exploited in our framework to further improve efficiency.

Traditional Indexes. A vast collection of index structures has been introduced in traditional databases, as well as Big Data systems (see e.g., M-trees [7]). Traditional spatial indexes, such as the R-tree family and kd-trees, are designed to improve the evaluation of a variety of spatial queries and are widely available in both disk-based and main memory implementations. Due to their objective (i.e., support of various spatial query operations), even main memory spatial indexes require substantial memory and time resources to construct [10], which makes them inappropriate for enabling the users to quickly start exploring and interacting with the data, as in the case of in situ data exploration (see also the results in Sect. 5). On the contrary, our approach proposes a main-memory lightweight index, which aims at accelerating the raw data-to-visualization time and offering a simple set of 2D visual operations to the user, rather than covering all aspects of spatial data management.

7 Conclusions

In this paper, we have presented the RawVis framework and the VALINOR index, a lightweight main memory structure, which enables interactive 2D visual exploration scenarios of very large raw data files in the presence of limited memory resources. VALINOR is constructed from a raw data file given the first user query and adapted based on the user interaction. We have formulated a set of simple visual operations and mapped them to query operators evaluated on the VALINOR index. We have conducted a thorough experimental evaluation with real and synthetic datasets and compared with three competitors; i.e., MySQL, PostgresRaw, and R-tree. The results showed that our technique outperforms both in query execution time and memory consumption.

Acknowledgments. This research is implemented through the Operational Program “Human Resources Development, Education and Lifelong Learning” and is co-financed by the European Union (European Social Fund) and Greek national funds.

References

1. Alagiannis, I., Borovica, R., Branco, M., Idreos, S., Ailamaki, A.: NoDB: efficient query execution on raw data files. In: SIGMOD (2012)
2. Battle, L., Chang, R., Stonebraker, M.: Dynamic prefetching of data tiles for interactive visualization. In: SIGMOD 2016 (2016)
3. Bikakis, N., Liagouris, J., Krommyda, M., Papastefanatos, G., Sellis, T.: GraphVizdb: a scalable platform for interactive large graph visualization. In: ICDE (2016)
4. Bikakis, N., Papastefanatos, G., Skourla, M., Sellis, T.: A hierarchical aggregation framework for efficient multilevel visual exploration and analysis. *Semant. Web J.* **8**, 139–179 (2017)
5. Blanas, S., Wu, K., Byna, S., Dong, B., Shoshani, A.: Parallel data analysis directly on scientific file formats. In: SIGMOD (2014)
6. Cheng, Y., Rusu, F.: SCANRAW: a database meta-operator for parallel in-situ processing and loading. *ACM Trans. Database Syst.* **40**(3), 1–45 (2015)
7. Ciaccia, P., Patella, M., Zezula, P.: M-tree: an efficient access method for similarity search in metric spaces. In: VLDB (1997)
8. de Lara Pahins, C.A., Stephens, S.A., Scheidegger, C., Comba, J.L.D.: Hashed-cubes: simple, low memory, real-time visual exploration of big data. *TVCG* **23**(1), 671–680 (2017)
9. El-Hindi, M., Zhao, Z., Binnig, C., Kraska, T.: VisTrees: fast indexes for interactive data exploration. In: HILDA (2016)
10. Hwang, S., Kwon, K., Cha, S.K., Lee, B.S.: Performance evaluation of main-memory R-tree variants. In: Hadzilacos, T., Manolopoulos, Y., Roddick, J., Theodoridis, Y. (eds.) SSTD 2003. LNCS, vol. 2750, pp. 10–27. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45072-6_2
11. Idreos, S., Alagiannis, I., Johnson, R., Ailamaki, A.: Here are my data files. Here are my queries. Where are my results? In: CIDR (2011)
12. Ivanova, M., Kersten, M.L., Manegold, S., Kargin, Y.: Data vaults database technology for scientific file repositories. *Comput. Sci. Eng.* **15**(3), 32–42 (2013)
13. Jugel, U., Jerzak, Z., Hackenbroich, G., Markl, V.: VDDA: automatic visualization-driven data aggregation in relational databases. *VLDBJ* **25**, 53–77 (2015)
14. Kalinin, A., Çetintemel, U., Zdonik, S.B.: Interactive data exploration using semantic windows. In: SIGMOD (2014)
15. Karpathiotakis, M., Branco, M., Alagiannis, I., Ailamaki, A.: Adaptive query processing on raw data. *PVLDB* **7**(12), 1119–1130 (2014)
16. Olma, M., Karpathiotakis, M., Alagiannis, I., Athanassoulis, M., Ailamaki, A.: Slalom: coasting through raw data via adaptive partitioning and indexing. *PVLDB* **10**(10), 1106–1117 (2017)
17. Tian, Y., Alagiannis, I., Liarou, E., Ailamaki, A., Michiardi, P., Vukolic, M.: DiNoDB: an interactive-speed query engine for ad-hoc queries on temporary data. *IEEE TBD* (2017)

Data Mining and Knowledge Discovery



Extended Margin and Soft Balanced Strategies in Active Learning

Dávid Papp^(✉) and Gábor Szűcs

Department of Telecommunications and Media Informatics,
Budapest University of Technology and Economics,
Magyar Tudósok krt. 2, Budapest 1117, Hungary
{pappd, szucs}@tmit.bme.hu

Abstract. Nowadays active learning is gaining increasing interest in computer vision community, especially on images. The most commonly used query strategy framework is uncertainty sampling usually in a pool-based sampling scenario. In this paper we propose two query strategies for image classification under the uncertainty sampling framework, both of them being improvements of existing techniques. The first strategy, so called Extended Margin incorporates all possible class labels to calculate the informativeness values of unlabeled instances. The second strategy is the improvement of the recently published BAL method, so called Soft Balanced approach, where we suggest new final informativeness score from an uncertainty measure and a novel penalty metric. We used least margin criterion for the former and the latter was calculated from the categorical penalty scores by using soft assignment. We conducted experiments on 60 different test image sets, each of them was a randomly selected subset of the Caltech101 image collection. The experiments were performed in an extended active learning environment and the results showed that the Extended Margin outperforms the least margin approach and the Soft Balanced method overcomes all other competitor method.

Keywords: Active learning · Query strategies · Uncertainty sampling
Image classification · Penalty metric · Soft assignment

1 Introduction

Active learning is a way of addressing a common problem, where unlabeled data may be easily obtained, but labels are expensive, time-consuming and difficult to create. For example consider such a movie annotation, where the target group consists of hearing impaired people. In this case accurate labeling of every sound (not only the speech) would be required and this is an extremely time consuming task. Another example could be the classification of different plant species based on visual information (photos), which requires trained biologists. In these scenarios the goal is to use as few labeled instances as possible, while retaining the same level of accuracy that could be achievable by using the total dataset. The method of active learning approaches this issue by giving the opportunity to the learning system to iteratively select which unlabeled instances to label. The key idea here is that if we allow the learner to query

the label of the presumably most informative instance, then it will be able to set up a more accurate model than with the label of a randomly selected instance.

There are three different problem scenarios that have been considered in the literature: membership query synthesis [8], stream-based selective sampling [9, 10] and pool-based sampling [11]. In this paper we only consider the pool-based sampling, in which we assume that a large collection of unlabeled data (U) is available at once. As can be seen in Fig. 1, the learner may ask queries in form of one or more carefully selected unlabeled instances to be labeled by an oracle (e.g. human annotator), then these new labeled instances are added to the labeled set (L). Based on the extended knowledge provided by the new labeled instances, the learner sets up a new machine learning model, then it estimates the informativeness values of the remaining unlabeled instances, and then it chooses which instances to query next.

One of the most important parts of an active learning algorithm is how it estimates the informativeness of unlabeled instances. There are many proposed query strategy frameworks in the literature, e.g. uncertainty sampling [36], query-by-committee (QBC) [34], expected model change [5], expected error reduction [22], to mention only the most commonly used ones. We developed and implemented two methods to predict the informativeness of the unlabeled instances, both of them belong to the uncertainty sampling framework.

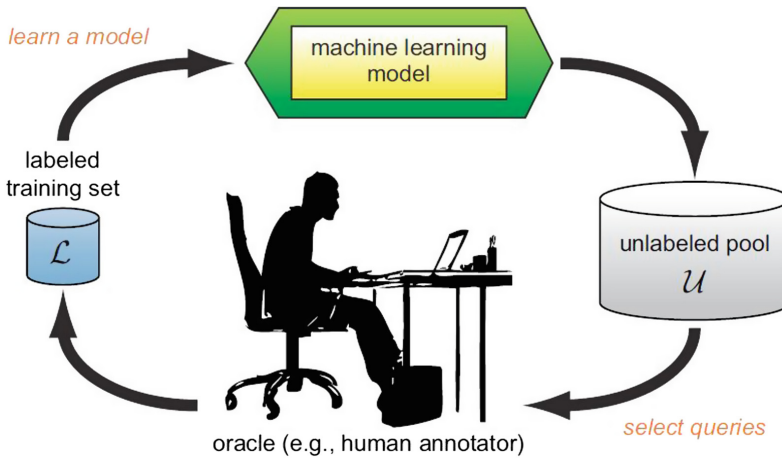


Fig. 1. Active learning cycle [2]

The rest of this paper is organized as follows. First we summarize the related literature in this area, then in Sect. 3 we present our proposed approaches, the Extended Margin and the Soft Balanced query strategies. In the next section we briefly review our solution for image classification. Section 5 presents the experimental results that were conducted on 60 different image datasets (each of them being a random subset of the Caltech101 image collection [19]), finally in the last one we describe our conclusion.

2 Related Work

We will focus on active learning problems especially on pool-based scenario and uncertainty sampling. The authors of [31] used a support vector active learning algorithm for conducting effective relevance feedback for image retrieval. In another work [4], an active learning framework was proposed to guide hidden annotations to improve retrieval performance in content-based information retrieval (CBIR). The authors of [35] used simple margin selection method for SVM for the active selection of object windows in images. The system autonomously refines its models by actively requesting crowd-sourced annotations on images crawled from the Web. On the other hand, in [25], an uncertainty sampling method was applied to image recognition in an active learning scenario; a margin sampling criterion and entropy criterion were used in the conditional part of this method. We compared our approaches with random sampling and the basic uncertainty sampling using least margin criterion in this paper.

Many other works on uncertainty sampling methods are based on the entropy notion. For example, in [36] an entropy criterion was used to measure the uncertainty of random variables in random walks on a graph. The authors of [12] present a so-called class balanced active learning (CBAL) framework for classifier training to detect cancerous regions on prostate histopathology. They also address the minority class problem, where one class is underrepresented, but their solution is only for binary problems. Another solution called Balanced Active Learning (BAL) method [26] applies a novel penalty metric to support entropy based uncertainty sampling, and balances the distribution of the already labeled instances among the categories (multi-class). We present an improvement of this method and compare it to the original version of BAL method.

One of the most recent paper [32] attempts to distinguish between the two types of uncertainties (conflicting-evidence vs. insufficient-evidence), but it does not provide another alternative approach for improving uncertainty sampling.

3 Proposed Query Strategies

3.1 Uncertainty Sampling

Uncertainty sampling is the most commonly used and probably the simplest query strategy framework, and its goal is to query the instance with least certainty about its true label [2]. Therefore, we measure the amount of uncertainty as informativeness value in this case. The easiest way is to select the image whose prediction is least confident, i.e. the maximum element of its decision vector is minimal (see Eq. 1).

$$x^* = \underset{x}{\operatorname{argmax}}(1 - P_M(y^*|x)) \quad (1)$$

where y^* denotes the class label with the highest probability under the model M . Despite its simplicity this method is very efficient, however it only uses the information provided by the most probable class label. Another uncertainty sampling variant called least margin calculates the difference between the first and second most

probable class labels, and then it selects the image whose difference value (margin) is minimal, as can be seen in Eq. 2.

$$x^* = \underset{x}{\operatorname{argmin}} (P_M(y_1^*|x) - P_M(y_2^*|x)) \quad (2)$$

where y_1^* and y_2^* denote the first and second most probable class labels, respectively. However, the least margin approach still ignores much of the remaining label distribution.

3.2 Extended Margin

To correct for this, we developed so-called Extended Margin technique, which incorporates all class labels into the formula. The informativeness value of an image is the weighted sum of $C - 1$ component margins, where C is the number of classes. Component i is the difference between the first and i^{th} most probable class labels. We calculate the Extended Margin metric for each unlabeled instance, and then we select the image corresponding to the lowest score; as can be seen in Eq. 3.

$$x^* = \underset{x}{\operatorname{argmin}} \sum_{i=2}^c w_{(i-1)} (P_M(y_1^*|x) - P_M(y_i^*|x)) \quad (3)$$

where $w_{(i-1)}$ represents the weight of component $i - 1$. We investigated several possible weight vectors (w), which were the following:

- $w = \{1, 1, 1, 1, \dots\}$
- $w = \{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \dots\}$
- $w = \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \dots\}$

The first, equal weights vector, gives too much influence to less important components. Component i is more important than component $i + 1$, because probability i is higher than probability $i + 1$, and therefore it is more likely that the classifier confuses the most probable class with class i than with class $i + 1$. In the second case we chose decreasing weights to follow the decreasing importance, moreover the value of the elements of the third weight vector is decreasing even faster, and we used this weight vector in our experiments. Furthermore, the Extended Margin can be considered as a general margin formula with arbitrary weights; e.g. if we choose $w = \{1, 0, 0, 0, \dots\}$ as weight vector then we get the least margin formula.

3.3 Soft Balanced

In this section we present the second proposed active learning strategy, so called Soft Balanced approach. The basis of our approach is the BAL method introduced in [26]. This kind of informativeness measurement consist of two parts, an uncertainty value and a so-called penalty metric. In the Soft Balanced strategy we used least margin criterion (see Eq. 2) for uncertainty measurement and we propose a new penalty metric defined as

$$PEN_i = CTR_i \times \frac{1}{C} \quad (4)$$

where $i = 1 \dots C$ and CTR_i denotes a general counter whose value increases by 1 with each iteration of queries when the received category is other than i . Note that we used categorical penalty metrics instead of giving each unlabeled instance its own. The individual penalty metric of an instance is composed from the categorical ones by taking their weighted average, where the weights are the corresponding class membership probabilities (as can be seen at the denominator of Eq. 5). We call this calculation method soft assignment, hence the “soft” prefix in the name of the method. The original BAL method uses hard assignment, which means that the penalty metric of an unlabeled instance is the penalty metric of the most probable class.

We normalized the CTR values by the number of classes (C), so the penalty metric also depends on the number of classes. When the learner queries an instance from a particular category, the CTR value (and therefore the penalty metric) of that category is reset to zero. Higher penalty metric represents higher uncertainty, and because of this an unlabeled instance with higher assigned penalty metric is more desirable to query than one with a lower value. Consequently, the learning system would query the instance with maximum penalty metric, while it would query the instance with minimum least margin criterion score. We merged these measurement scores into a final decision score by taking the ratio of them, as can be seen in the following equation:

$$S_{x_j} = \frac{P_M(y_1^*|x_j) - P_M(y_2^*|x_j)}{\sum_{i=1}^C (P_M(y_i|x_j) \times PEN_i)} \quad (5)$$

where the nominator represents the least margin score of instance x_j similarly to Eq. 2. In the denominator we calculate the penalty metric of instance x_j by taking the categorical penalty metrics over all possible labeling. It is important to mention that the denominator is always a positive number, and therefore we did not have to handle the case of division with zero. S_{x_j} denotes the final uncertainty score of instance x_j , and the algorithm aims to query the unlabeled instance corresponding to the minimal S . The advantage of our proposed approach is that it establishes a balance among the classes of labeled instances.

In order to test our active learning query strategy methods in image classification, in each iteration we had to calculate the class membership probabilities ($P_M(y_i|x_j)$) used in the above equations; in the next section we describe it in details.

4 Image Classification

In this section we will briefly review the image representation and image classification processes; we applied BoV (Bag-of-Visual-Words) model [13, 18, 20] for the former and SVM (Support Vector Machine) [1, 7, 18] for the latter. The key idea behind BoV model is to represent an image based on its visual content with so-called visual code

words while ignoring their spatial distribution. This technique consists of three steps, these being (i) feature detection, (ii) feature description and (iii) image description as usual phases in computer vision. For feature detection we utilized the Harris-Laplace corner detector [3, 24], and then we used SIFT (Scale Invariant Feature Transform) [21] to extract and describe the local attributes of the features. Note that we used the default parameterization of SIFT proposed by Lowe; hence we got descriptor vectors with 128 dimensions. To define the visual code words from the descriptor vectors, we used the GMM (Gaussian Mixture Model) [30, 33], which is a parametric probability density function represented as a weighted sum of K Gaussian component densities; as can be seen in Eq. 6.

$$p(X|\lambda) = \sum_{j=1}^K \omega_j g(X|\mu_j, \sigma_j) \quad (6)$$

where X is the concatenation of all SIFT descriptors, ω_j , μ_j and σ_j denote the weight, expected value and variance of the j^{th} Gaussian component, respectively; and $K = 256$. We calculated the λ parameter, which includes the parameters of all Gaussian functions, with ML (Maximum Likelihood) estimation by using the iterative EM (Expectation Maximization) algorithm [16, 33]. We performed K-means clustering [23] over all the descriptors with $K = 256$ clusters to get the initial parameter model for the EM. The next step was to create a descriptor that specifies the distribution of the visual code words in an image, called high-level descriptor. To represent an image with high-level descriptor, the GMM based Fisher-vector [14, 28] was calculated. This is an appropriate and complex descriptor vector, because this is able to take the semantic essence of the picture, and this is already validated in classification problems [14, 15, 27, 28]. The Fisher-vector is computed from the SIFT descriptors of an image based on the visual code words by taking the derivative of the logarithmic of the Gaussian functions (see Eq. 7), thus it describes the distribution of the visual elements for an image. These vectors were the final representation of the images, and we used them as input for the classification.

$$FV = \nabla_{\lambda} \log p(X|\lambda) \quad (7)$$

where $p(X|\lambda)$ is the probability density function introduced in Eq. 1, X denotes the SIFT descriptors of an image and λ represents the parameter of GMM ($\lambda = \{\omega_j, \mu_j, \sigma_j | j = 1 \dots K\}$).

For the classification subtask we used a variation of SVM, the C-SVC (C-support vector classification) with RBF (Radial Basis Function) kernel. The one-against-all technique was applied to extend the SVM for multi-class classification. We used Platt's [29] approach as probability estimator, which is included in LIBSVM (A Library for Support Vector Machines [6, 17]). We used these algorithms in each iteration of the active learning cycle to calculate the class membership probabilities that were used to estimate the informativeness of the unlabeled instances.

5 Experimental Results

5.1 Experimental Environment

We used the Caltech101 [19] image collection for conducting our experiments, which consists of 8677 images from 101 different categories. More precisely, we used 60 different, randomly selected and possibly overlapping subsets of Caltech101 as test image sets. To get these subsets we performed the following operations 20 times for each different category number (C):

1. Randomly selected 30 images from each category
2. Divided each category into train and test parts (20 images in train, 10 images in test)
3. Randomly selected $C = \{5, 10, 20\}$ classes from the total 101 classes.

We repeated the selection 20 times for each different category number to be able to take the average of the results got on them; and therefore to give a comprehensive result and conclusion about what is expected if one chooses one of the given number of classes. In the rest of the paper we will consider only the image set types, instead of the individual image sets one by one; also later on, when we present the results of the image set types, we mean the averaged results got on the 20 individuals.

It is important to mention that we used an extended environment for our experiments, which means that we had an additional fix test image set (F) on which we evaluated the results (as can be seen in Fig. 2). This is the reason why we divided the images into train and test parts; the test part in each category will be the fix image set. However, we need to classify the unlabeled pool (U), because we calculate the informativeness values based on the decision vectors. We could also evaluate the results got on U , but it would be misleading, especially in case of uncertainty sampling, since we query an instance that is difficult to classify; thus eliminating a probably misclassification is further increasing the accuracy in the new iteration (aside from the new label). Thereby we chose to use a fix test image set to measure the accuracy.

The labeled training set (L) is a very small set in the beginning, we randomly choose an image from each category at the start, and then in each iteration it is expanded by one more image and its label. The iteration stops when L and U has the same size, and at that point L and F also has the same size. Note that by same size we mean equal number of images.

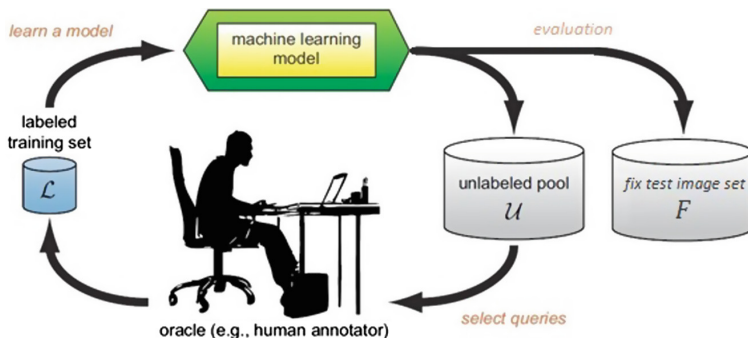


Fig. 2. Extended active learning cycle

We evaluated six strategies for comparison, these were:

- Random sampling (RS)
- Uncertainty sampling using least margin (LM)
- Original BAL method (using entropy criterion, BAL1)
- BAL method using least margin criterion (BAL2)
- Extended Margin (ExM)
- Soft Balanced sampling using least margin (SBAL).

5.2 Results

Now, we will present our experimental results that were conducted on the 60 test image sets. We call the image set types with 5, 10 and 20 categories by Caltech5, Caltech10 and Caltech20, respectively; and we present the results got on them in Figs. 3, 4 and 5. Each figure consists of two graphs where the left one shows the results of RS, LM, BAL1 and BAL2 methods represented as long dashed, dashed, dotted and solid lines, respectively; while the right one shows the results of BAL2, ExM and SBAL methods represented as dashed, dotted and solid lines, respectively. Both graphs have the accuracy on the y-axis and the number of trained images on the x-axis. We separated the methods to make the figures less crowded and easier to read. As can be seen in the left side of the figures, random sampling gave the lowest accuracy almost at all points on each image set type. The uncertainty sampling with least margin criterion gave much higher accuracies than RS, and the original BAL method was a little bit better than LM. Furthermore, the results obtained by using BAL1 and BAL2 methods show that we were able to create a better query strategy by (only) changing the uncertainty measurement technique inside the original BAL method from entropy to least margin.

On the right graph of the figures we compare our proposed query strategies only to BAL2, since it gave the highest accuracy among the competitor methods. Therefore, if we show that ExM and SBAL outperform BAL2, then it implies that they also outperform RS, LM and BAL1. As can be seen in the figures, SBAL gave the highest accuracy at almost all points, while ExM gave higher accuracy than LM at each point. In fact, ExM gave similar accuracies as BAL1 and in some cases it even outperformed BAL2.

We summarize the results of the experiments in Table 1, where we use abbreviations to encode the image set type and the number of trained images; e.g. ‘C5 #50’ represents the results obtained on Caltech5 image set type with 50 training images in the labeled pool. Basically, Table 1 includes the accuracies got on each image set type with each tested methods after querying one quarter or half of the unlabeled pool. Furthermore, we calculated the standard deviation of the accuracies because the experiments were performed 20 times on each image set type. We denote the rows that contain the standard deviation values with ‘sdev’ and the ones that contain average accuracy values with ‘acc’. Based on the results we may conclude that ExM is a better uncertainty sampling criterion than LM and SBAL is the best query strategy among the investigated methods.

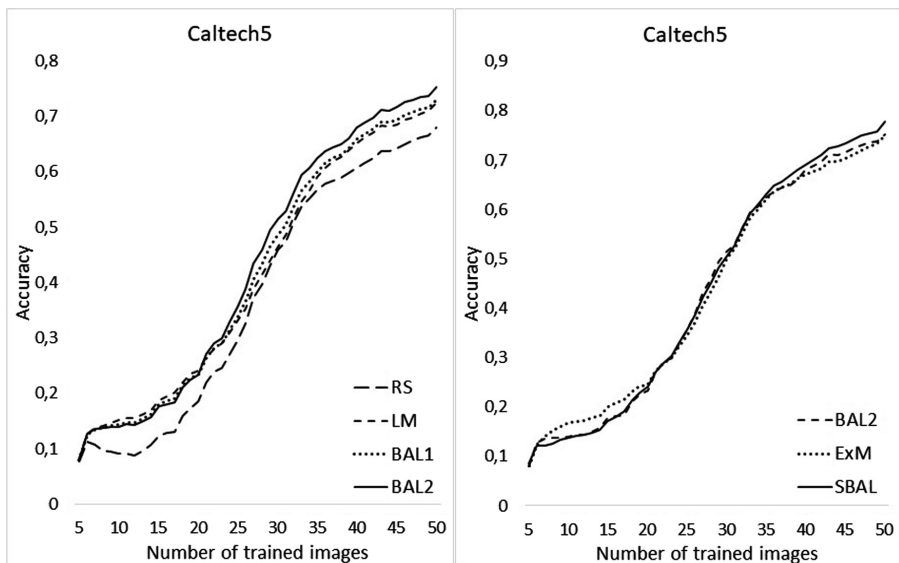


Fig. 3. Results got on Caltech5 image set type

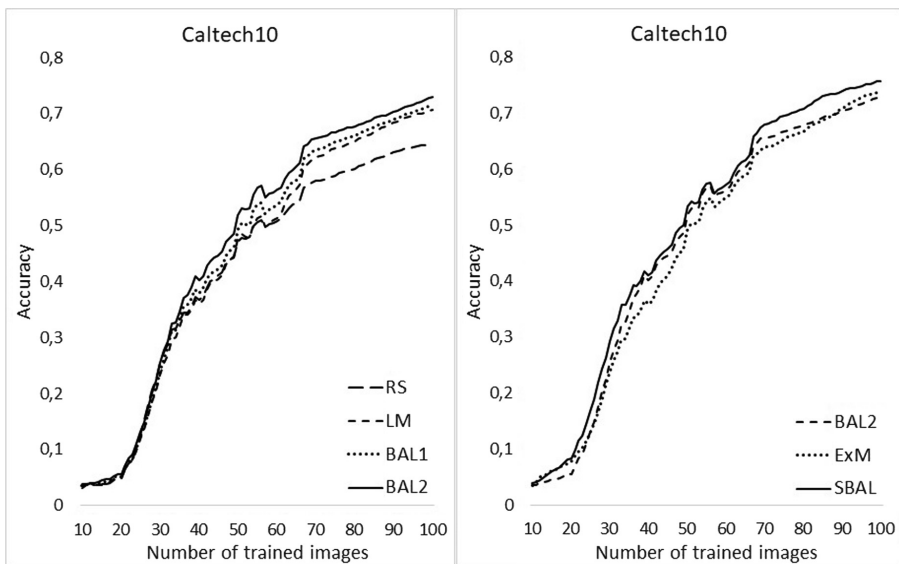


Fig. 4. Results got on Caltech10 image set type

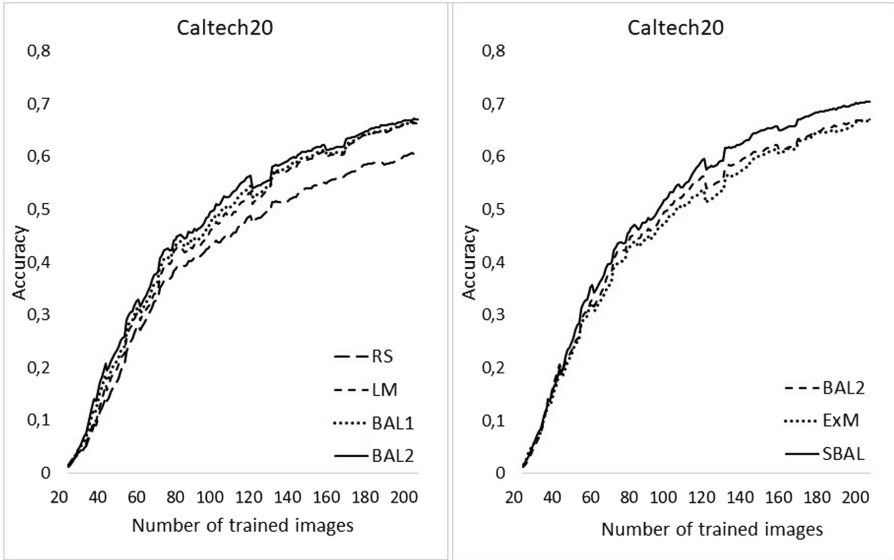


Fig. 5. Results got on Caltech20 image set type

Table 1. Average and standard deviation of the accuracy values evaluated on each image set type with each tested method

	RS	LM	BAL1	BAL2	ExM	SBAL
C5 #25 acc	0.295	0.330	0.340	0.356	0.344	0.357
C5 #50 acc	0.679	0.724	0.730	0.751	0.747	0.778
C5 #25 sdev	0.174	0.167	0.026	0.186	0.143	0.156
C5 #50 sdev	0.119	0.126	0.075	0.146	0.142	0.125
C10 #50 acc	0.470	0.477	0.495	0.519	0.490	0.533
C10 #100 acc	0.645	0.707	0.714	0.729	0.738	0.757
C10 #50 sdev	0.075	0.090	0.041	0.085	0.104	0.093
C10 #100 sdev	0.060	0.070	0.039	0.069	0.076	0.067
C20 #100 acc	0.446	0.489	0.504	0.524	0.504	0.545
C20 #200 acc	0.604	0.663	0.664	0.671	0.668	0.704
C20 #100 sdev	0.054	0.056	0.021	0.045	0.064	0.057
C20 #200 sdev	0.039	0.048	0.027	0.047	0.050	0.051

6 Conclusion

In this paper we presented two new query strategies for active learning in a pool-based scenario. Both of these approaches belong to the uncertainty sampling framework because they measure the uncertainty among the possible labels as informativeness of the unlabeled instances. The proposed Extended Margin (ExM) is the general case of

least margin and it incorporates all class labels into formula. On the other hand, our Soft Balanced method (SBAL) uses categorical penalty metrics to determine the individual penalties of the unlabeled instances and it merges these scores with the ones coming from an uncertainty measurement with least margin criterion. We tested the proposed query strategies on 60 randomly selected subsets of Caltech101 image collection and compared the obtained results to several other uncertainty sampling methods. Our experiments showed that SBAL outperforms all of the competitor methods, while ExM gave higher classification accuracy than the basic least margin during the tests, moreover many times it gave the second best accuracy behind SBAL.

Acknowledgement. The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013).

References

1. Boser, B., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifier. In: Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, pp. 144–152 (1992)
2. Settles, B.: Active learning. *Synth. Lect. Artif. Intell. Mach. Learn.* **6**(1), 1–114 (2012)
3. Harris, C., Stephens, M.: A combined corner and edge detector. In: Proceedings of the Alvey Vision Conference, pp. 23.1–23.6 (1988)
4. Zhang, C., Chen, T.: An active learning framework for content based information retrieval. *IEEE Trans. Multimed.* **4**(2), 260–268 (2002)
5. Cai, W., Zhang, Y., Zhou, J.: Maximizing expected model change for active learning in regression. In: IEEE 13th International Conference on Data Mining, pp. 51–60 (2013)
6. Chang, C.-C., Lin, C.-J.: LIBSVM: a library for support vector machines. *ACM Trans. Intell. Syst. Technol.* **2**(3), 27.1–27.27 (2011)
7. Cortes, C., Vapnik, V.: Support-vector networks. *Mach. Learn.* **20**(3), 273–297 (1995)
8. Angluin, D.: Queries and concept learning. *Mach. Learn.* **2**, 319–342 (1988)
9. Cohn, D., Atlas, L., Ladner, R.: Improving generalization with active learning. *Mach. Learn.* **15**(2), 201–221 (1994)
10. Cohn, D., et al.: Training connectionist networks with queries and selective sampling. In: Advances in Neural Information Processing Systems (NIPS). Morgan Kaufmann, Burlington (1990)
11. Lewis, D., Gale, W.: A sequential algorithm for training text classifiers. In: Croft, B.W., van Rijsbergen, C.J. (eds.) SIGIR 1994, pp. 3–12. Springer, Heidelberg (1994). https://doi.org/10.1007/978-1-4471-2099-5_1
12. Doyle, S., Monaco, J., Feldman, M., Tomaszewski, J., Madabhushi, A.: A class balanced active learning scheme that accounts for minority class problems: applications to histopathology. In: OPTIMHisE Workshop (MICCAI), pp. 19–30 (2009)
13. Fei-Fei, L., Fergus, R., Torrvalba, A.: Recognizing and learning object categories. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2007)
14. Perronnin, F., Dance, C.: Fisher kernel on visual vocabularies for image categorization. In: Computer Vision and Pattern Recognition (CVPR) (2007)
15. Gosselin, P.H., Murray, N., Jégou, H., Perronnin, F.: Revisiting the fisher vector for fine-grained classification. *Pattern Recogn. Lett.* **49**, 92–98 (2014)

16. Gupta, M.R., Chen, Y.: Theory and use of the EM algorithm. *Sig. Process.* **4**(3), 223–296 (2010)
17. Huang, T.-K., Weng, R.C., Lin, C.-J.: Generalized Bradley-Terry models and multi-class probability estimates. *J. Mach. Learn. Res.* **7**, 85–115 (2006)
18. Chatfield, K., Lempitsky, V., Vedaldi, A., Zisserman, A.: The devil is in the details: an evaluation of recent feature encoding methods. In: *Proceedings of the 22nd British Machine Vision Conference*, pp. 76.1–76.12 (2011)
19. Fei-Fei, L., Fergus, R., Perona, P.: Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Workshop on Generative-Model Based Vision (2004)*
20. Lazebnik, S., Schmid, C., Ponce, J.: Beyond bags of features: spatial pyramid matching for recognizing natural scene categories. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 2169–2178 (2006)
21. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vis.* **60**(2), 91–110 (2004)
22. Mac Aodha, O., Campbell, N., Kautz, J., Brostow, G.: Hierarchical sub-query evaluation for active learning on a graph. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 564–571 (2014)
23. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281–297 (1967)
24. Mikolajczyk, K., Schmid, C.: Scale & affine invariant interest point detectors. *Int. J. Comput. Vis.* **60**(1), 63–86 (2004)
25. Minakawa, M., Raytchev, B., Tamaki, T., Kaneda, K.: Image sequence recognition with active learning using uncertainty sampling. In: *Proceedings of the International Joint Conference on Neural Networks*, pp. 1–6 (2013)
26. Papp, D., Szűcs, G.: Balanced active learning method for image classification. *Acta Cybernetica* **23**(2), 645–658 (2017)
27. Perronnin, F., Liu, Y., Sánchez, J., Poirier, H.: Large-scale image retrieval with compressed fisher vectors. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3384–3391 (2010)
28. Perronnin, F., Sánchez, J., Mensink, T.: Improving the fisher kernel for large-scale image classification. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) *ECCV 2010. LNCS*, vol. 6314, pp. 143–156. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15561-1_11
29. Platt, J.: Probabilistic outputs for support vector machines and comparison to regularize likelihood methods. In: *Advances in Large Margin Classifiers*, pp. 61–74 (2000)
30. Reynolds, D.A.: Gaussian mixture models. In: *Encyclopedia of Biometric Recognition*, pp. 659–663 (2009)
31. Tong, S., Chang, E.: Support vector machine active learning for image retrieval. In: *Proceedings of the ACM International Conference on Multimedia*, pp. 107–118 (2001)
32. Sharma, M., Bilgic, M.: Evidence-based uncertainty sampling for active learning. *Data Min. Knowl. Disc.* **31**, 164 (2017)
33. Tomasi, C.: Estimating Gaussian mixture densities with EM - a tutorial. Technical report, Duke University (2004)

34. Tsai, Y.L., Tsai, R.T.H., Chueh, C.H., Chang, S.C.: Cross-domain opinion word identification with query-by-committee active learning. In: Cheng, S.M., Day, M.Y. (eds.) TAAI 2014. LNCS, vol. 8916, pp. 334–343. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-319-13987-6_31
35. Vijayanarasimhan, S., Grauman, K.: Large-scale live active learning: training object detectors with crawled data and crowds. *Int. J. Comput. Vis.* **108**(1–2), 97–114 (2014)
36. Yang, Y., Ma, Z., Nie, F., Chang, X., Hauptmann, A.G.: Multi-class active learning by uncertainty sampling with diversity maximization. *Int. J. Comput. Vis.* **113**(2), 113–127 (2015)



Location-Awareness in Time Series Compression

Xu Teng^{1(✉)}, Andreas Züfle^{2(✉)}, Goce Trajcevski^{1(✉)}, and Diego Klabjan³

¹ Department of Electrical and Computer Engineering, Iowa State University,
Ames, IA, USA

{xuteng,gocet25}@iastate.edu

² Department of Geography and Geoinformation Science, George Mason University,
Fairfax, VA, USA

azufle@gmu.edu

³ Department of Industrial Engineering, Northwestern University, Evanston, IL, USA
d-klabjan@northwestern.edu

Abstract. We present our initial findings regarding the problem of the impact that time series compression may have on similarity-queries, in the settings in which the elements of the dataset are accompanied with additional contexts. Broadly, the main objective of any data compression approach is to provide a more compact (i.e., smaller size) representation of a given original dataset. However, as has been observed in the large body of works on compression of spatial data, applying a particular algorithm “blindly” may yield outcomes that defy the intuitive expectations – e.g., distorting certain topological relationships that exist in the “raw” data [7]. In this study, we quantify this distortion by defining a measure of similarity distortion based on Kendall’s τ . We evaluate this measure, and the correspondingly achieved compression ratio for the five most commonly used time series compression algorithms and the three most common time series similarity measures. We report some of our observations here, along with the discussion of the possible broader impacts and the challenges that we plan to address in the future.

1 Introduction and Motivation

Modern advances in sensing technologies – e.g., weather stations, satellite imagery, ground and aerial LIDAR, weather radar, and citizen-supplied observation – have enabled representing the physical world with high resolution and fidelity. The trend of *Next Generation Sensor Networks and Environmental Science* [9] aims at integrating various data sources (e.g., offered by the state-of-the-art GEOS-5 data assimilation system [26]) and make them publicly available. An example of such large scale dataset is the MERRA-2 data, provided

X. Teng—Research supported by NSF grant III 1823267.

G. Trajcevski—Research supported by NSF grants III-1823279 and CNS-1823267, and ONR grant N00014-14-1-0215.

by NASA [19] – covering the whole time period of the modern era of remotely sensed data, from 1979 until today, and recording a large variety of environmental parameters, e.g., temperature, humidity and precipitation; on a spatial resolution of 0.5° latitude times 0.67° longitude produced at one-hour intervals. This, in turn, enables access to many Terabytes of historic evolution in time of environmental data.

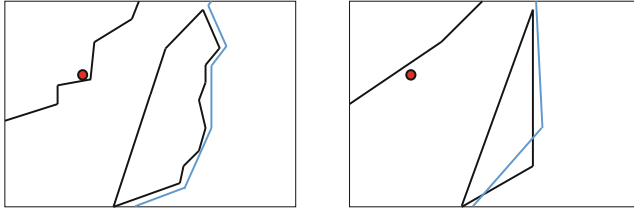


Fig. 1. Simplification and topological distortions (based on [7])

Although the focus of this work is on the peculiarities of compressing time series and the interplay with other contexts, to better understand the motivation we briefly turn the attention to compression in spatial data. In the mid 1990s, concurrently with the advances in cartography and maps management [30], the multitude of application domains depending on geographic properties (e.g., distributions) of various phenomena in agriculture, health, demographics, etc. [11], brought about the field of Spatial databases [24]. Most of the compression techniques applied in spatial datasets rely on some kind of a *line simplification* approach, and many variants have also been extensively studied by the Computational Geometry (CG) community [3, 28]. Among of the most popular line simplification approaches is Douglas-Peucker¹ (DP) [6]. However, as demonstrated in [20], applying the DP algorithm to reduce the polylines bounding the polygons in a given subdivision, may often cause topological inconsistencies, as illustrated in Fig. 1, in the following sense:

- Boundaries of regions which were not intersecting in the original representation may end up intersecting after the simplification is applied. Similarly, the simplified polylines corresponding to different regions may intersect each other.
- Relative position of point-locations with respect to a boundary or a polyline may change after the simplification is applied – e.g., a city which was on the north bank of the river may end up in its south bank after the polyline representing the river has been simplified.

¹ Around the same time, there were other algorithms developed for polyline simplification, some of which had almost-identical methodologies with the DP algorithm. Most notably [16] which is the reason that sometimes the name Ramer-Douglas-Peucker is used in the literature.

One of the canonical problems in time series is the *similarity search* – i.e., given a collection/database of time series and a particular query-sequence, detect which particular time series is most similar to the querying one, with respect to a given distance function [5]. Since time series databases are large in size, much research has been devoted to speeding up the search process. Among the better known and used paradigms are the ones based on techniques that perform dimensionality reduction on the data, which enables the use of spatial access methods to index the data in the transformed space [14]. Many similarity measurements and distance functions for time series have been introduced in the literature [5] – however, what motivates our work is rooted at the observation that large datasets that are time series by nature, are often tied with other context attributes. Sources of such time series exist in many different domains – such as location-aware social networks [8,31] and atmospheric and precipitation data [21,23] (but two examples).

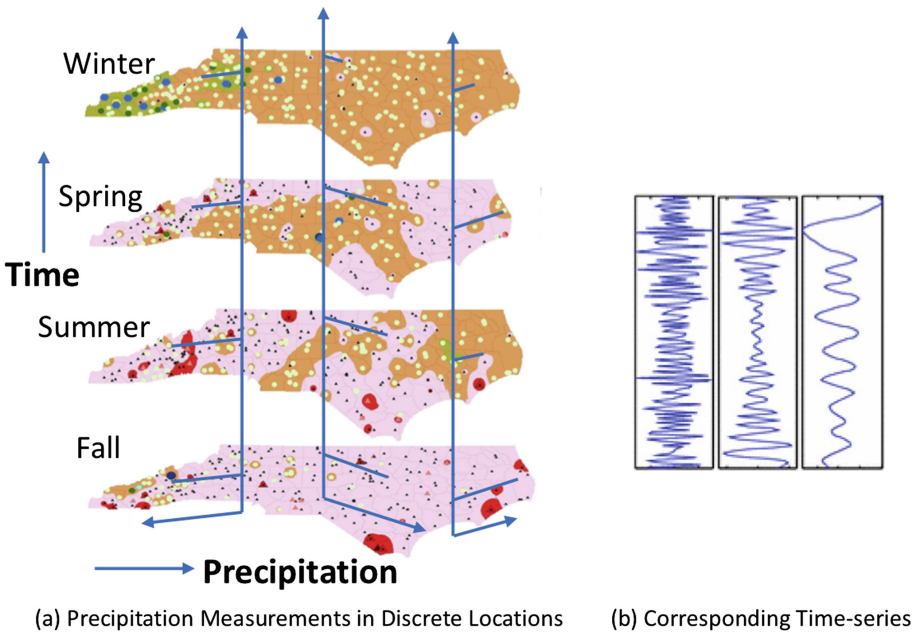


Fig. 2. Precipitation time series

Our key observations are:

- O_1 : Given the size of such datasets, one would naturally prefer to store the data in a compressed/simplified representation.
- O_2 : Many queries of interest over such datasets may involve values from >1 context/domain.

For example, Fig. 2(a) (cf. [17, 21]) illustrates the *spatial* distribution of the measurements of precipitation in discrete locations. However, in each individual location, the collection of the measurements from different time-instants actually form a time series – as illustrated in Fig. 2(b) which shows the detailed corresponding precipitation time series. In the spirit of O_1 and O_2 above, consider the following query:

Q_1 : Which location in the continental US has the most similar distribution of monthly precipitation with Ames?

The main motivation for this work is to investigate the impact of different compression approaches on variability of the answer(s) to Q_1 above. While in the case of Q_1 the additional context is the *location*, we postulate that other queries pertaining to time series with additional contexts may suffer from distortion of their answers. Such distortions, in turn, may affect the choice of a particular compression algorithm to be used – e.g., as part of materializing the data in dimensions-hierarchy of warehouses [27]. In this work, we report our initial findings in this realm.

In the rest of the paper, Sect. 2 defines the problem settings and Sect. 3 reviews the compression approaches and respective measures. In Sect. 4 we discuss in detail our observations to date, and in Sect. 5 we summarize and outline directions for future work.

2 Problem Definition

In broadest terms, *data compression* can be perceived as a science or an art – or a mix of both – aiming at development of efficient methodologies for a compact representation of information [10, 22]. Information needs a representation – be it a plain text file, numeric descriptors of images/video, social networks, etc. – and one can rely on properties of structure, semantics, or other statistically-valid features of that representation when developing the methodologies for making the underlying representation more compact. Speaking a tad more formally, *data compression* can be defined as any methodology that can take a dataset D with a size β bits as an input, and produce a dataset D' as a representation of D and having a size β' bits, where $\beta' < \beta$ (hopefully, $\beta' \ll \beta$).

To measure the capability of a data compression algorithm to reduce the size of a dataset, in this work we simply rely on the typical measure – the compression ratio [10].

Definition 1 (Compression Ratio). Let D be a dataset represented by β_D bits. Let \mathcal{C} be a compression function, which maps D to a compressed dataset $\mathcal{C}(D)$ represented by $\beta_{\mathcal{C}(D)}$ bits. We define the compression ratio of \mathcal{C} on D as:

$$\mathcal{R}_{\mathcal{C}}(D) = \frac{\beta_D}{\beta_{\mathcal{C}(D)}}.$$

We note that the representation size β_D is not necessarily equal to the entropy $E(D)$ of D [18]: The entropy of D is the smallest possible number of bits required to represent D . Thus, it must hold that $E(D) \leq \beta_D$. The aim of this study is not to evaluate the information aspects of time series theoretically, but rather, to see the impact of the loss of a particular type of information incurred by compression algorithm on practical queries related to similarity search on time series.

Clearly, one can easily find a compression algorithm that maximizes the compression ratio of Definition 1, by a “brute force” discarding any and all information. However, such an approach would inhibit any meaningful similarity search among the compressed time series, as all of them would be equally-valid candidates for an answer. Thus, the challenge approached in this work is to maximize the compression ratio while maintaining similarity search results as accurate as possible.

To measure how a compression algorithm \mathcal{C} can maintain similarity search results among a set D of time series tied with other context attributes, we compute similarity rankings between all the time series. A similarity ranking, using a query time series $T \in D$, ranks all other time series in $D \setminus T$ by their similarity to T . To quantify the similarity ranking before vs. after the compression, we employ Kendall’s rank correlation coefficient τ [12], which measures how many pairs of relative ranking positions are preserved and discordant between the two rankings. Formally,

Definition 2 (Ranking Similarity). *Let D be a set of time series. For a query time series $T \in D$, let $\text{Rank}(T, D)$ be the similarity ranking of T to all other time series $T' \in D \setminus T$. Further, let \mathcal{C} be a compression algorithm, let $\mathcal{C}(D)$ denote the compressed representation of D , and let $\text{Rank}(T, \mathcal{C}(D))$ denote the similarity ranking of T after the compression. Then, we describe the similarity of these two rankings as:*

$$\tau(\text{Rank}(T, D), \text{Rank}(T, \mathcal{C}(D))) = \sum_{T_i, T_j \in D, i < j} \frac{I(\text{conc}(T_i, T_j)) - I(\text{disc}(T_i, T_j))}{(|D|^2 - |D|)/2},$$

where either $(I(\text{conc}(T_i, T_j)))$ or $(I(\text{disc}(T_i, T_j)))$ is an indicator function that returns 1 if time series T_i and T_j are concordant or discordant in both rankings (that is, if the relative ranking order between T_i and T_j is maintained or not in both rankings) and 0 otherwise.

As an example, consider a case where we have four time series T_1, \dots, T_4 , and assume that the similarity ranking of T_1 is (T_2, T_3, T_4) , implying that T_2 is most similar to T_1 , while T_4 is the least similar one. Further, assume that after compression, the ranking becomes (T_2, T_4, T_3) . In this case, the relative order between T_2 and T_3 is preserved, as is the relative order between T_2 and T_4 . The only “discordant” order is between T_3 and T_4 , yielding $\tau((T_2, T_3, T_4), (T_2, T_4, T_3)) = \frac{1}{3}$.

To quantify the overall information maintained between all of the time series, we compute the average τ score of all time series in D .

Definition 3 (Average Ranking Similarity). Let D be a set of time series and let $\mathcal{C}(D)$ denote the compressed representation of D . We define the average ranking similarity between D and $\mathcal{C}(D)$ as

$$\tau(D, \mathcal{C}(D)) = \frac{\sum_{T \in D} \tau(\text{Rank}(T, D), \text{Rank}(T, \mathcal{C}(D)))}{|D|}.$$

We reiterate that our goal is to evaluate how different compression algorithms \mathcal{C} affect the balance between compression ratio (Definition 1) and average ranking similarity (Definition 3).

3 Compressions and Distances

For self-containment, we now briefly survey the compression techniques and distance measures used in this study.

3.1 Compression Approaches

We have used two broad categories of compression techniques, as described in detail in the sequel.

Dimensionality Reduction. Instead of being viewed as a collection of n time-instant phenomena, a time series, $\{t_1, t_2, \dots, t_n\}$, can be considered as a point in n -dimensional space. Dimensionality reduction approaches focus on reducing the dimensionality – from n in the “native”, to m ($m < n$) in the lower dimensional space – while minimizing the loss of explained variance. We use two representative techniques:

- *Discrete Fourier Transform:*

The key idea of Discrete Fourier Transform (DFT) [2] is based on the observation that any n -length time series can be represented in the frequency domain with n sine and cosine waves, that can be used to reconstruct the original time series. The compression stems from the observation that the waves with low amplitudes can be neglected without losing too much valuable information.

- *Piecewise Aggregate Approximation:*

The basic concept behind the Piecewise Aggregate Approximation (PAA) [14] is dividing the original time series into N equally sized windows, where (N is the desired dimensionality of transformed space). Then, each window/frame is represented by the mean value of all the data within that particular frame. The formula used for performing PAA on an n -dimensional time series and transforming it into the N -dimensional space is shown in Eq. 1:

$$\bar{t}_i = \frac{N}{n} \sum_{j=\frac{N}{n}(i-1)+1}^{\frac{N}{n}i} t_j, \quad i = 1, 2, \dots, N \quad (1)$$

One may observe that a small window-size can achieve a better performance on preserving information, but yields a poor compression ratio – e.g., when the window size is equal to n , the transformed representation is identical to the original time series.

Native-Space Compression. Another kind of compression approaches reduces the size of the initial time series in its “native space”:

- *(Adapted) Douglas-Peucker Algorithm:*
 Given a sequence of time series and a user-defined tolerance threshold ε , the Douglas-Peucker (DP) [6] algorithm recursively sub-divides the input sequence based on an “anchor”. An “anchor” is a point that has a largest distance exceeding ε from the line segment connecting the initiator (first point initially) and the terminus (last point initially). The DP algorithm is traditionally used to compress polylines. To adapt it to time series, we use vertical (instead of perpendicular) distance in this study. Vertical distance between point t_k and line segment (t_i, t_j) , $i < k < j$, is defined as $|t'_k - t_k|$, where t'_k is the intersection of line segment (t_i, t_j) and the line passing t_k and perpendicular to the time-axis.
- *Visvalingam-Whyatt Algorithm:*
 The key aspect of Visvalingam-Whyatt (VW) [29] algorithm is the “effective area”, which indicates the surface area of the triangle formed by a point with its two neighbors. For a time series of length n , a total $(n-2)$ triangles can be formed. The main idea behind the VW algorithm is to iteratively drop the middle point of the triangle with the least “effective area” and keep on updating the triangles related to that displaced point until the “effective area” is larger than the user-given parameter ε .
- *(Adapted) Optimal Algorithm:*
 The main idea of optimal algorithm (OPT) [4] is to consider two directions (*forward* and *backward*), for each point of a time series. For instance, $(t_{i+1}, t_{i+2}, \dots, t_n)$ is *forward* for t_i , and $(t_{i-1}, t_{i-2}, \dots, t_1)$ is *backward*. The i -th ($1 \leq i \leq n$) pass of the algorithm draws circles with radius ε , centered at each the *forwards* and *backward* points of t_i – denoted *Circle* $_{i+1}$, *Circle* $_{i+2}$, ..., *Circle* $_n$ and *Circle* $_{i-1}$, *Circle* $_{i-2}$, ..., *Circle* $_1$. Take *forward* chain as instance. While touching a new point, t_k , $i < k \leq n$, let U_k and L_k indicate the upper and the lower ray emanating from t_i , passing through the top and bottom point of *Circle* $_k$ – in a sense, defining a wedge pertaining to t_k and with the apex at t_i . For as long as the intersection of successive wedges is not empty, nothing needs to be updated except of recording the lowest-upper and highest-lower boundary of the intersection maintained so far. Otherwise, denote t_k as the *event point* which generates an empty intersection. We keep t_i and t_{k-1} into the result and repeat the procedure from the event point t_{k-1} forwards. Similarly for the *backward* chain of t_i .

3.2 Distance Measures

Existing literature has identified many scenarios where similarity cannot be simply evaluated by any single distance function [5]. Thus, for validity, we used three measurements in this work, as described next.

Pearson Correlation Coefficient. The Pearson product-moment correlation coefficient [15] (denoted r) is a widely used lock-step measure for relationship. By Cauchy-Schwartz inequality, the range of r is established to the interval $[-1, +1]$, where $+1$ denotes total positive linear correlation, 0 is no linear correlation, and -1 indicates negative linear correlation.

Dynamic Time Warping. Dynamic Time Warping (DTW) [13] is an elastic similarity measure between two temporal sequences. In general, it focuses on calculating an optimal match between two given time series that may vary in speed/frequency. Unlike lock-step methods, DTW alignment may match a point from one sequence to one or more points of another sequence.

Cosine Similarity. Cosine similarity [25] aims at evaluating the orientation difference between two time series, and is independent of the magnitude of the samples. If two sequences are with a same orientation, their cosine similarity will be 1; and if their orientation difference is 90° , then their similarity will be zero.

4 Experimental Observations

In this section, we present the experimental evaluations of the approaches discussed in Sect. 3 in terms of compression rate and average ranking similarity. Our data sets are obtained from the University Corporation for Atmospheric Research (UCAR) and the National Center for Atmospheric Research (NCAR) at the Global Precipitation Climatology Centre [1].

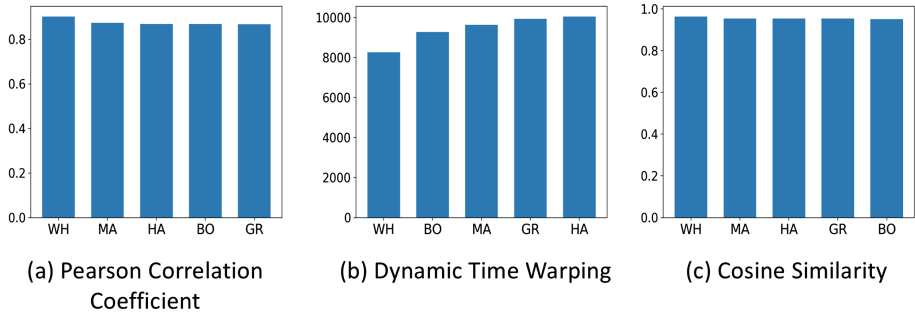


Fig. 3. Location-based similarity scores

Table 1. Locations used in the reported experiments

Location	Abbreviation	Location	Abbreviation
Wheeler, IA	WH	Massena, IA	MA
Hartford, IA	HA	Boyer, IA	BO
Grant, IA	GR	Garfield, IA	GA
South Kidder, ND	SK	Union County, NM	UC
Otter Creek, IA	OC	Courtland Township, MN	CT
Anoka County, MN	AC	Grant Township, SD	GT
Grantsburg, WI	GB	Scott, WI	SC
Hazelhurst, WI	HH		

Recall the motivational question stated in Sect. 1: *Which location in the continental US has most similar distribution of monthly precipitation with Ames?* In this spirit, we extract 50 years worth of monthly precipitation data for Ames and other 500 land areas in the United States. Figure 3 shows the top five locations having highest similarity with Ames measured by Pearson Correlation Coefficient, DTW and Cosine Similarity, respectively. The horizontal axis shows the abbreviation of each locations, and the corresponding full name can be found at Table 1. The vertical axis shows the similarity score of each locations. As discussed in Sect. 3, higher score means better performance for Pearson Correlation Coefficient and Cosine Similarity, and DTW pursues lower distance. We can figure out that the five locations listed in Fig. 3(a), (b) and (c) are same though the ranking has some differences.

Figure 4 states the effect of two Dimensionality Reduction methods on ranking. 0.7 in terms of multiples of the maximum value of each time series is defined

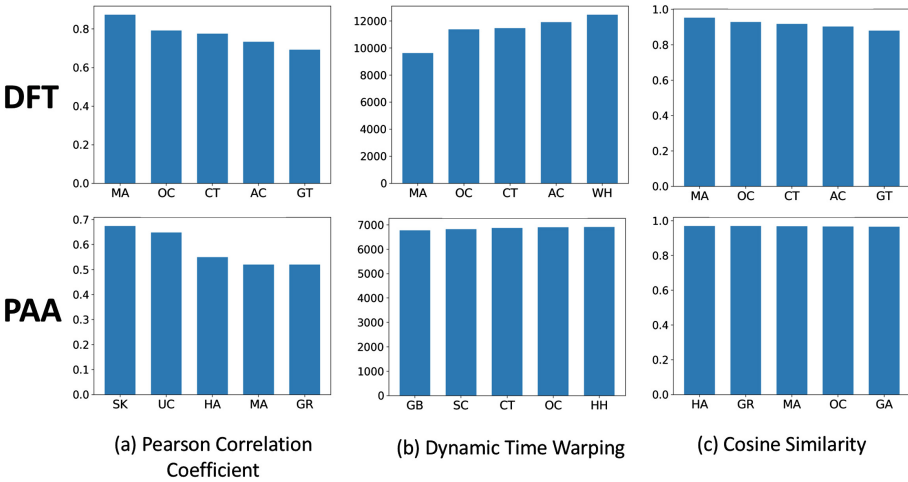


Fig. 4. Dimensionality reduction compression

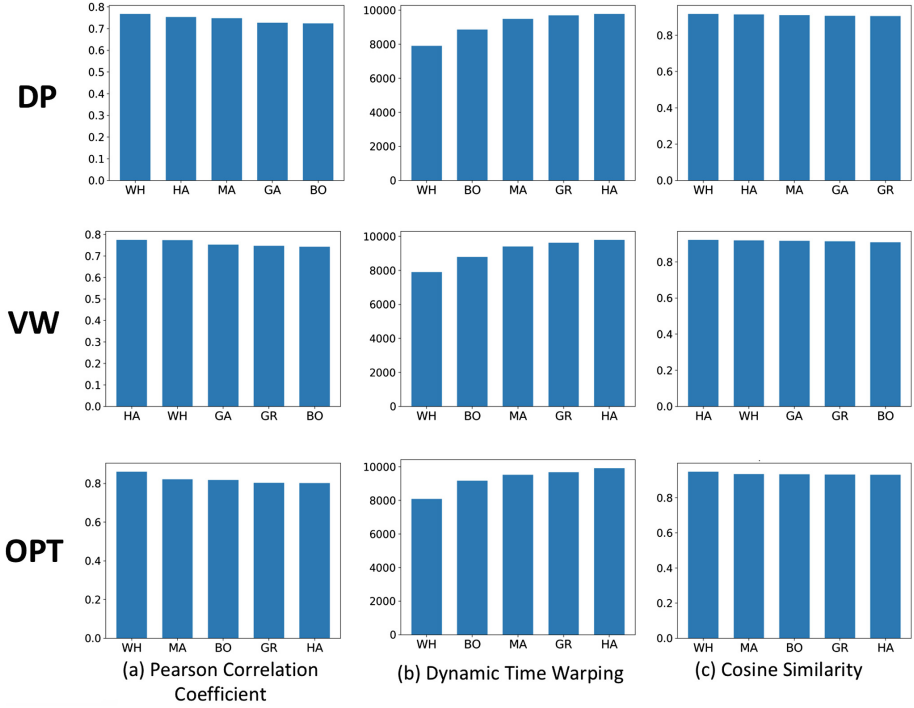


Fig. 5. Native-space compression

as *error tolerances*. We can discover the only half of the locations have no difference with Fig. 3.

Figure 5 illustrates the influence of three different Native-Space compression approaches mentioned in Sect. 3 on ranking of similarity. For DP and VW approaches, the *error tolerances* are set to be 5. For the OPT algorithm, the values of tolerance are set to be the half of those of DP and VM algorithms. As can be seen from results, though the compression do influence the ranking of top five locations, all the locations in the result are the same as the ranking of ground truth except of Grant, IA. And performance is similar to Fig. 3 and better than Fig. 4 while checking the similarity score.

To evaluate the influence of different compression approaches on time series in a more general way, we randomly samples 100 land areas in the United States, and fetched 50 years of monthly precipitation data for each location in our work. Similarly to the above experiment, for each selected area, we can get three different rankings of similarity across the rest of 99 locations measured by Pearson Correlation Coefficient, DTW and Cosine Similarity. As mentioned in Sect. 2, the goal is to evaluate the influence of different compression approaches on compression ratio and impacts on the average ranking similarity. In order to perform such comparison, we set the single parameter *error tolerance* for

different algorithms. For DFT and PAA approaches, the *error tolerances* are set to be 0.7, 0.75, 0.8, 0.85, 0.9, 1 in terms of multiples of the maximum value of each dataset. For DP and VW approaches, the *error tolerances* are 5, 10, 15, 20, 25, 30. Lastly, for the OPT algorithm, the values of tolerance are set to be the half of those of DP and VM algorithms.

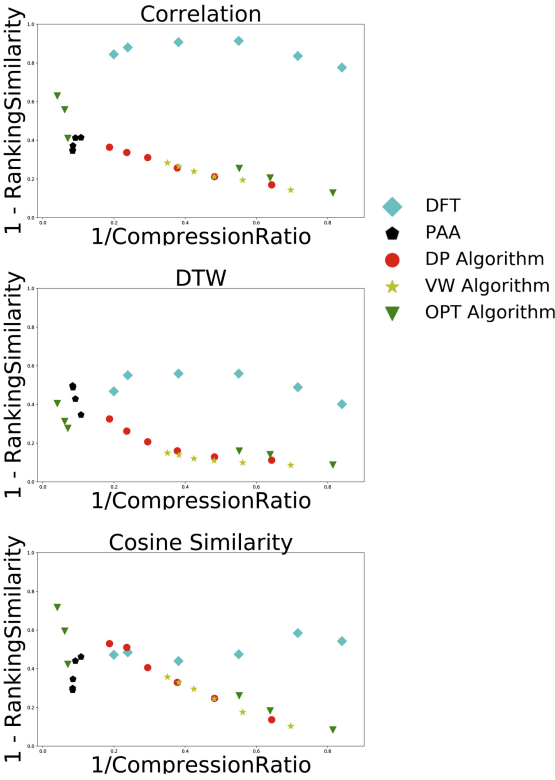


Fig. 6. Global similarity distortions

comparison with the other measures. Thus, we conclude that even when achieving a fairly low compression rate, DFT loses most of information required to maintain the original similarity ranking. This loss is not compensated by additional frequency features, as these low amplitude features mostly incur additional noise.

Secondly, we observe that PAA achieves a relatively high compression rate, but the averaging of consecutive time stamps yields a high loss of information that drops τ to roughly 60% for all the other applied distance functions.

In contrast, we observe that all three native-space compression algorithms have comparable performances. We note that the OPT algorithm generally achieves worse results, and is dominated by DP and VW – which, in part, may be a consequence of setting a lower error-threshold. We observe minor difference

Figure 6 illustrates one more of our experimental observations that we now discuss. Note that, in a sense, we have transformed the coordinates and the respective values are: — the x-axis for each of the three graphs represents $1/\mathcal{R}_C(D)$; — the y-axis in each of the three graphs represents $1 - \tau(D, \mathcal{C}(D))$. This transformation was made to ensure that a “good” approach is close to the origin of the chart. For instance, a point at the (0,0) origin would correspond to a perfect τ of 1, and a perfect compression rate of ∞ .

Firstly, we observe that there exists no single approach that clearly dominates all the other approaches, in terms of both τ -score and compression rate. However, we also observe that DFT performs rather poorly in

between DP and VW, expect when DTW is used a distance measure: namely, VW is able to maintain more ranking for the similarity-based semantics in the DTW space.

We note that, for reproducibility, the source code for all the implementations used in our experiments, along with the corresponding dataset, is publicly available².

5 Summary and Future Directions

Satellites and other sensory devices have enabled a generation of extremely large environmental time series datasets. Ultimately, this data has the potential to transform our understanding of the world for a plethora of applications of societal relevance, such as meteorology, agriculture, urban development, traffic management, etc. However, this understanding is hindered by the overwhelming deluge of $O(\text{Petabytes})$ of such data. To reduce this data, the state-of-the-art offers many time series compression algorithms.

In this study, we experimentally evaluated the trade-off between the data reduction and the loss of semantics when an additional context – location in this work – is associated with each time series. Rather than measuring the theoretical loss of entropy, we measured how the incurred distortion changes similarity search results on environmental time series, using precipitation time series as a case-study.

Our main experimental finding is that dimensionality reducing methods, such as Discrete Fourier Transform and Piecewise Aggregate Approximation incur a high loss of similarity between compressed time series, relative to the original ones. In contrast, native space compression algorithms obtain similar compression rates, but maintain much more of the similarity information between time series. In particular, the Visvalingam-Whyatt algorithm and the Douglas-Peucker algorithm yield the best trade-off. Moreover, when Dynamic Time Warping is used as a similarity metric, Visvalingam-Whyatt has a significant advantage over Douglas-Peucker.

Our main objectives for the future are: (1) extend this study to include more compression algorithms, and include different types of environmental time series other than precipitation; and (2) investigate the impact of compression on semantics of other context attributes – e.g., in addition to location, exploit the (joint) impact on other social networks features; (3) evaluate the potential impacts of running time of the algorithms, especially in the sense of updating the datasets from newly available observations.

Acknowledgments. We thank Praxxal Patel and Yash Thesia for their help in finalizing part of the experiments.

² <https://github.com/XuTengNU/ADBIS2018.git>.

References

1. GPCC: Global Precipitation Climatology Centre. <https://climatedataguide.ucar.edu/climate-data/gpcc-global-precipitation-climatology-centre>
2. Agrawal, R., Faloutsos, C., Swami, A.: Efficient similarity search in sequence databases. In: Lomet, D.B. (ed.) FODO 1993. LNCS, vol. 730, pp. 69–84. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57301-1_5
3. Barequet, G., Chen, D.Z., Daescu, O., Goodrich, M.T., Snoeyink, J.: Efficiently approximating polygonal paths in three and higher dimensions. *Algorithmica* **33**(2), 150–167 (2002)
4. Chan, W.S., Chin, F.: Approximation of polygonal curves with minimum number of line segments. *Int. J. Comput. Geom. Appl.* **6** (1992)
5. Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E.J.: Querying and mining of time series data: experimental comparison of representations and distance measures. *PVLDB* **1**(2) (2008)
6. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitised line or its caricature. *Can. Cartographer* **10**(2) (1973)
7. Estkowski, R., Mitchell, J.S.B.: Simplifying a polygonal subdivision while keeping it simple. In: Symposium on Computational Geometry (2001)
8. Ferrari, L., Rosi, A., Mamei, M., Zambonelli, F.: Extracting urban patterns from location-based social networks. In: Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Location-Based Social Networks, LBSN 2011 (2011)
9. Hey, T., Tansley, S., Tolle, K.M., et al.: The Fourth Paradigm: Data-Intensive Scientific Discovery, vol. 1. Microsoft Research, Redmond (2009)
10. Hirschberg, D., Lelewer, D.A.: Data compression. *Comput. Surv.* **19**(3) (1987)
11. Kasperson, J.X., Kasperson, R.E., Turner II, B.L.: *Regions at Risk: Comparisons of Threatened Environments*. United Nations University Press, Tokyo (1995)
12. Kendall, M.G.: A new measure of rank correlation. *Biometrika* **30**, 81–93 (1938)
13. Keogh, E., Ratanamahatana, C.A.: Exact indexing of dynamic time warping. *Knowl. Inf. Syst.* **7**, 358–386 (2005)
14. Keogh, E.J., Chakrabarti, K., Pazzani, M.J., Mehrotra, S.: Dimensionality reduction for fast similarity search in large time series databases. *Knowl. Inf. Syst.* **3**(3) (2001)
15. Pearson, K.: Note on regression and inheritance in the case of two parents. *Proc. Roy. Soc. London* (1895)
16. Ramer, U.: An iterative procedure for the polygonal approximation of plane curves. *Comput. Graph. Image Process.* **1**, 244–256 (1972)
17. Reddy, M.J., Adarsh, S.: Time-frequency characterization of sub-divisional scale seasonal rainfall in India using the Hilbert-Huang transform. *Stochast. Environ. Res. Risk Assess.* **30**, 1063–1085 (2016)
18. Rényi, A.: On measures of entropy and information. Technical report, Hungarian Academy of Sciences, Budapest, Hungary (1961)
19. Rienecker, M.M., et al.: Merra: Nasa’s modern-era retrospective analysis for research and applications. *J. Climate* **24**(14), 3624–3648 (2011)
20. Saalfeld, A.: Topologically consistent line simplification with the Douglas-Peucker algorithm. *Cartography Geogr. Inf. Sci.* **26**(1), 7–18 (1999)
21. Sayemuzzaman, M., Jha, M.K.: Seasonal and annual precipitation time series trend analysis in North Carolina, United States. *Atmos. Res.* **137** (2014)

22. Sayood, K.: Introduction to Data Compression. Morgan Kauffman, Burlington (1996)
23. Sharma, C.S., Panda, S.N., Pradhan, R.P., Singh, A., Kawamura, A.: Precipitation and temperature changes in Eastern India by multiple trend detection methods. *Atmos. Res.* **180** (2016)
24. Shekhar, S., Chawla, S.: Spatial Databases: A Tour. Prentice Hall, Upper Saddle River (2003)
25. Steinbach, M., Karypis, G., Kumar, V., et al.: A comparison of document clustering techniques. In: *KDD Workshop on Text Mining*, vol. 400, pp. 525–526 (2000)
26. Suarez, M.J., et al.: The geos-5 data assimilation system-documentation of versions 5.0. 1, 5.1. 0, and 5.2. 0 (2008)
27. Vaisman, A.A., Zimányi, E.: Data Warehouse Systems - Design and Implementation. *Data-Centric Systems and Applications*. Springer, Heidelberg (2014). <https://doi.org/10.1007/978-3-642-54655-6>
28. van Kreveld, M., Luo, J.: The definition and computation of trajectory and sub-trajectory similarity. In: *GIS* (2007)
29. Visvalingam, M., Whyatt, J.D.: Line generalisation by repeated elimination of points. *Cartographic J.* **30**(1) (1993)
30. Weibel, R.: Generalization of spatial data: principles and selected algorithms. In: van Kreveld, M., Nievergelt, J., Roos, T., Widmayer, P. (eds.) *CISM School 1996*. LNCS, vol. 1340, pp. 99–152. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63818-0_5
31. Yang, G., Züfle, A.: Spatio-temporal prediction of social connections. In: *Proceedings of the Fourth International ACM Workshop on Managing and Mining Enriched Geo-Spatial Data*, Chicago, IL, USA, 14 May 2017 (2017)

Indexing, Query Processing and Optimization



Efficient SPARQL Evaluation on Stratified RDF Data with Meta-data

Flavio Ferrarotti¹(✉), Senén González¹, and Klaus-Dieter Schewe²

¹ Software Competence Center Hagenberg, Hagenberg, Austria
{flavio.ferrarotti, senen.gonzalez}@scch.at

² Christian-Doppler Laboratory for Client-Centric Cloud Computing, Linz, Austria
kdschewe@acm.org

Abstract. The Resource Description Framework (RDF) is a simple, but frequently used W3C standard, which uses triplets to define relationships between resources. In this paper the evaluation of queries in the query language SPARQL on RDF data with meta-data is investigated. We first show that if the data are stratified, i.e. a particular partial order can be defined on the meta-data labels, then a nesting procedure can be applied, which induces a rewriting of the query. Based on a specification by an Abstract State Machine we show that the result of the rewritten query equals the one that would have resulted from the evaluation of the original query. We further investigate the reduction of complexity by using data and query nesting.

Keywords: RDF · SPARQL · Meta-data · Nesting

1 Introduction

The Resource Description Framework (RDF) is a simple, but frequently used W3C standard [4, 9], which uses triplets to define relationships between resources. A triplet (s, p, o) is a statement with the informal meaning that resource s (the subject of the statement) is linked to resource o (the object of the statement) via information in the resource p (the predicate of the statement). Resources are identified by a unique identifier, for which the language of Internationalised Resources Identifiers (IRI) is commonly used. Instead of using an IRI in subject or object, an auxiliary identifier (commonly referred to as blank node) can be used. For the object it is further possible that an arbitrary value, referred to as label, can be used. When we talk about RDF data, we usually mean a set of such RDF triplets.

SPARQL is a common pattern-based query language for RDF data [12]. In a nutshell, a pattern is also a triplet, but may contain variables. By matching

The research reported in this paper was partially supported by the **Austrian Science Fund (FWF)[I2420-N31]** for the project: *Higher-Order Logics and Structures*.

patterns with RDF triplets we obtain bindings for the variables. A query consists of a set of patterns that must be simultaneously matched, and a specification of the output variables. As observed among others by Hartig [7], the writing and evaluation of SPARQL queries can become very cumbersome, when meta-data is to be exploited, e.g. concerning information about the creator or source of a triplet. The meta-data itself requires already at least three additional triplets for each triplet in RDF data and patterns in queries have to refer to all of these. Alternatives are the use of a singleton property [11], the application of named graphs [10] or emergent schemata [13]. Meta-data represented in this way also thwart common methods for SPARQL query optimisation [14] and provenance [5]. In order to solve this problem, the idea explored in [8] is to first rewrite both the RDF data and the queries using nested triplets, i.e. to permit triplets to appear as subject, object or both in RDF statements, and then to evaluate the rewritten query on the rewritten data.

The purpose of this paper is to provide a thorough investigation of such a process, which is quite common in the optimised evaluation of complex database queries (see for instance [1]). Our contributions are a simplification of the conditions, under which nesting of RDF data and query rewriting is possible, proofs of the correctness of such a process, and a thorough analysis of the efficiency gain. Using a specification by an Abstract State Machine (ASM) [2] we show how to represent RDF data and SPARQL queries and how to specify the transformation into the nested versions, provided there are no cycles among the meta-data, for which we provide a stratification check, which is also specified by an ASM. Our stratification condition is much simpler than the nesting condition used in [8] as well as in earlier works on nested RDF such as [3]. Then we proceed with analysing the nested evaluation procedure in two directions emphasising correctness and complexity.

In Sect. 2 we start with a general introduction to RDF, meta-data and SPARQL, which leads us to the definition of nested data and consequently nested SPARQL queries. In Sect. 3 we then show how nested SPARQL queries can be evaluated for which we introduce ASM specifications. Section 4 contains a formal analysis of the approach, first showing the correctness of the specifications, in particular the preservation of query result under the nesting procedure, second showing the reduction of complexity by using data and query nesting.

2 Stratified RDF Data and SPARQL

In this section we briefly introduce RDF and SPARQL, and provide a characterisation, under which conditions RDF data and SPARQL queries can be nested.

2.1 RDF Data and Meta-data

Definition 2.1. Let \mathcal{I} , \mathcal{A} and \mathcal{V} be pairwise disjoint sets of resource identifiers, auxiliary labels, and values, respectively. An *RDF triplet* is a triplet (s, p, o) with $p \in \mathcal{I}$, $s \in \mathcal{I} \cup \mathcal{A}$ and $o \in \mathcal{I} \cup \mathcal{A} \cup \mathcal{V}$. A finite set \mathcal{R} of RDF triplets is simply referred to as *RDF data*.

Example 2.1. Statements about movies can be represented by RDF triplets such as (:hitchcock, :director, :psycho), (:hitchcock, :name, “Hitchcock”), (:hitchcock, :firstname, “Alfred”), (:psycho, :genre, :thriller), and (:psycho, :year, “1960”) where we marked resource identifiers in \mathcal{I} by a leading colon, and values in \mathcal{V} by quotes.

In the simplistic RDF model the representation of meta-data, i.e. data about statements, e.g. their source, creator and the like, is rather cumbersome. In order to integrate meta-data in RDF data auxiliary labels can be used in combination with reserved resource identifiers such as :subject, :predicate and :object.

Example 2.2. We may add triplets ($_s$, :subject, :hitchcock), ($_s$, :predicate, :director) and ($_s$, :object, :psycho) using an auxiliary label $_s \in \mathcal{A}$ to represent the first statement in the previous example. Then we can add statements such as ($_s$, :source, “http://www.imdb.com/title/tt0054215/”) to state that the information that Alfred Hitchcock directed the movie Psycho can be found on the given web page. Here we should guarantee that the additional labels such as $_s$ are only linked to a unique statement.

The introduction of meta-data in this way may lead to awkward RDF data. For instance, assume triplets ($_s_2$, :p, :o₂) represented by the auxiliary label $_s_1$, and ($_s_1$, :p, :o₁) represented by the auxiliary label $_s_2$, which introduces an undesirable cycle.

As discussed in detail in the RDF literature (see e.g. [7]) the writing and evaluation of queries on RDF data can become complicated, when meta-data as in Example 2.2 is to be exploited. Each statement requires at least three additional triplets and all these triplets must be used in a query. Therefore, it is desirable to permit a representation of meta-data that does not require the addition of many auxiliary triplets. This can be achieved by nesting, i.e. extending RDF in such a way that triplets may appear directly within triplets.

Definition 2.2. A *nested RDF triplet* based on RDF data \mathcal{R} is either an RDF triplet in \mathcal{R} or a triplet (s, p, o) with $p \in \mathcal{I}$, $s \in \mathcal{I} \cup \mathcal{A} \cup \mathcal{R}^*$ and $o \in \mathcal{I} \cup \mathcal{A} \cup \mathcal{V} \cup \mathcal{R}^*$, where \mathcal{R}^* is the set of all nested RDF triplets based on \mathcal{R} .

Example 2.3. We can create nested RDF triplets ((:hitchcock, :director, :psycho), :source, “http://www.imdb.com/title/tt0054215/”) to represent the same meta-data as in Example 2.2.

However, if we are given RDF data with cycles, e.g. ($_s_2$, :p, :o₂) represented by the auxiliary label $_s_1$, and ($_s_1$, :p, :o₁) represented by the auxiliary label $_s_2$ —we omit the meta-data here—then the nested analogue becomes a rational tree (((...($_s_2$, :p, :o₂)...), :p, :o₁), :p, :o₂), which is hard to handle in queries, unless fixed-point constructions are supported.

In order to exclude cycles we require RDF data to be stratified. For this we assign stratum numbers $\text{strat}(\ell) \in \mathbb{N}$ to each label in $\mathcal{St} = \{\ell \in \mathcal{A} \mid \exists s, p, o. \text{represents}(\ell, (s, p, o))\}$ using the predicate represents defined by

$$\text{represents}(\ell, (s, p, o)) \equiv \mathcal{R}(\ell, :subject, s) \wedge \mathcal{R}(\ell, :predicate, p) \wedge \mathcal{R}(\ell, :object, o).$$

Definition 2.3. RDF data \mathcal{R} are called *stratified* iff there exists an assignment $\text{strat} : \mathcal{St} \rightarrow \mathbb{N}$ of stratum numbers such that $\text{strat}(\ell) > \text{strat}(\ell')$ holds, whenever $\text{represents}(\ell, (s, p, o))$ holds with $s = \ell' \vee o = \ell'$.

Example 2.4. The RDF data and meta-data in Examples 2.1 and 2.2 are trivially stratified with $\mathcal{St} = \{_{-}s\}$.

However, the cyclic RDF data in Example 2.3 are not stratified, as it is impossible to assign stratum numbers to $_{-}s_1$ and $_{-}s_2$. The condition in 2.3 would require $\text{strat}(_{-}s_1) < \text{strat}(_{-}s_2) < \text{strat}(_{-}s_1)$, which is impossible.

2.2 SPARQL Queries

SPARQL queries are conjunctive queries defined by patterns, which are simply triplets with variables that have to be matched to the given RDF data.

Definition 2.4. Let V be a set of variables. An *RDF pattern* is a triplet (s, p, o) with $p \in \mathcal{I} \cup V$, $s \in \mathcal{I} \cup \mathcal{A} \cup V$ and $o \in \mathcal{I} \cup \mathcal{A} \cup \mathcal{V} \cup V$, where \mathcal{I} , \mathcal{A} and \mathcal{V} are as in Definition 2.1.

A *query* is a pair (V_q, \mathcal{P}) with a finite set \mathcal{P} of RDF patterns and a set of result variables V_q that is a subset of the set of variables appearing in the patterns in \mathcal{P} .

Example 2.5. Consider the query $((\{\$m, \$y\}), \mathcal{P})$ with the patterns $(\$p, :director, \$m)$, $(\$p, :name, \text{“Hitchcock”})$, $(\$p, :firstname, \text{“Alfred”})$ and $(\$m, :year, \$y)$ in \mathcal{P} , based on RDF data in Example 2.1. Clearly the result will be a list of movies directed by Alfred Hitchcock together with the production year.

If in addition we take the meta-data in Example 2.2 into account, add $\$w$ to the result variables, and add the patterns $(\$s, :source, \$w)$, $(\$s, :subject, \$p)$, $(\$s, :predicate, :director)$ and $(\$s, :object, \$m)$ to \mathcal{P} , the query result will also contain the source of the information that a movie was directed by Hitchcock.

The semantics of a query is obvious. We look for all possible simultaneous instantiations of the patterns by labels matching the given RDF data and include the tuple of labels assigned to the result variables into the result relation, i.e.

$$\text{Result}(V_q, \mathcal{P}) = \{(a_{i_1}, \dots, a_{i_m}) \mid [a_1/x_1, \dots, a_n/x_n].p \in \mathcal{R} \text{ for all } p \in \mathcal{P}\}, \quad (2.1)$$

where $V_{\mathcal{P}} = \{x_1, \dots, x_n\}$ is the set of variables appearing in \mathcal{P} and $V_q = \{x_{i_1}, \dots, x_{i_m}\} \subseteq V_{\mathcal{P}}$.

If RDF data are nested in order to permit easier representation of meta-data as explained in the previous subsection, the evaluation procedure above will no longer produce the correct results. We have to extend queries to nested queries as well.

Definition 2.5. A *nested RDF pattern* is either an RDF pattern or a triplet (s, p, o) with $p \in \mathcal{I} \cup V$, $s \in \mathcal{I} \cup \mathcal{A} \cup \mathbb{P} \cup V$ and $o \in \mathcal{I} \cup \mathcal{A} \cup \mathcal{V} \cup \mathbb{P} \cup V$, where \mathbb{P} is the set of all nested RDF patterns over \mathcal{R} .

A *nested query* is a pair (V_q, \mathcal{P}) with a finite set \mathcal{P} of nested RDF patterns and a set of result variables V_q that is a subset of the set of variables appearing in the patterns in \mathcal{P} .

We preserve the definition of semantics in Eq. (2.1). In particular, variables will be bound to labels in $\mathcal{I} \cup \mathcal{A} \cup \mathcal{V}$, not to triplets. The only difference is that for a nested query \mathcal{R} represents nested RDF data.

Example 2.6. Consider again the second query in Example 2.5. On nested RDF data the nested query with the same result variables $V_q = \{\$m, \$y, \$w\}$ and the nested RDF patterns $((\$p, :director, \$m), :source, \$w)$, $(\$p, :name, \text{“Hitchcock”})$, $(\$p, :firstname, \text{“Alfred”})$ and $(\$m, :year, \$y)$ in \mathcal{P} will produce the desired result.

3 Evaluation of SPARQL on Stratified RDF Data

We now define the general procedure for the evaluation of SPARQL queries on RDF data with meta-data. Exploiting the idea of nesting we first have to check, whether the RDF data and meta-data are stratified, for which we define a procedure STRATIFIED. If this is the case, we first transform the RDF data into nested RDF data using a procedure NEST, then we transform the query into a nested query using a procedure NEST-QUERY. Finally, we use a procedure EVALUATE to evaluate the transformed query. In case the data is not stratified, the procedure EVALUATE will be applicable nonetheless, but there will be no gain in complexity reduction.

3.1 ASMs in a Nutshell

We will define an Abstract State Machine (ASM) [2] for this evaluation procedure. Then STRATIFIED, NEST, NEST-QUERY, and EVALUATE will become ASM rules. First we give a brief description of ASMs.

States in an ASM \mathcal{M} are Tarski structures, i.e. there exists a signature Σ associated with \mathcal{M} comprising a set of function symbols. Each function symbol $f \in \Sigma$ has an arity $ar(f) \in \mathbb{N}$. For a fixed set B of values (usually called base set) we obtain a *structure* by interpretation of the function symbols: a function symbol f of arity n gives rise to a function $f_S : B^n \rightarrow B$, and the state S is defined by these functions. The interpretation is extended to terms in the usual way. Note that a function symbol of arity 0 is usually called a variable, if it can be updated, or a constant, if it is static, i.e. its interpretation is never changed.

A pair (f, \bar{b}) consisting of a function symbol f of arity n and an n -tuple $\bar{b} = (b_1, \dots, b_n) \in B^n$ is called a *location*. A pair (ℓ, b) comprising a location ℓ and a value $b \in B$ is called an *update*.

Furthermore, \mathcal{M} has a rule, by means of which states can be updated. Rules are composed inductively using the following constructors:

assignment. $f(t_1, \dots, t_{ar_f}) := t_0$ with terms t_i built over Σ ,
branching. IF φ THEN r_+ ELSE r_- ENDIF (the ELSE-branch is optional),
parallel composition. FORALL x WITH $\varphi(x)$ DO $r(x)$ ENDDO,
bounded parallel composition. PARBLOCK $r_1 r_2 \dots r_n$ ENDPAR,
choice. CHOOSE x WITH $\varphi(x)$ DO $r(x)$ ENDDO

The informal meaning is as follows. An assignment evaluates the terms on the left hand side, which define an $ar(f)$ -tuple \bar{b} and thus a location $\ell = (f, \bar{b})$. The evaluation of the term on the right hand side defines a value b , which is to become the new value at location ℓ in a successor state. For branching the Boolean term φ is evaluated to either *true* or *false*, and then either r^+ or r^- is applied depending on this truth value. For parallel composition the Boolean term $\varphi(x)$ is evaluated, and for all values of x that lead to *true* the corresponding rules $r(x)$ are executed in parallel. For bounded parallel composition all rules r_1, \dots, r_n are executed in parallel. For choice the Boolean term $\varphi(x)$ is evaluated, and among those values of x that lead to *true* one is selected and the corresponding rule $r(x)$ is executed.

Formally, each rule r , when applied in a state S , yields an update set $\Delta_r(S)$ —we omit the rather obvious definition—and this update set is applied to state S to define a successor state $S + \Delta_r(S)$, where the value at a location $\ell = (f, \bar{b})$ is defined by

$$f_{S+\Delta_r(S)}(\bar{b}) = \begin{cases} b & \text{if } (\ell, b) \in \Delta_r(S) \\ f_S(\bar{b}) & \text{else} \end{cases}.$$

A *run* of an ASM \mathcal{M} with rule r is a sequence S_0, S_1, \dots of states, where S_0 is an initial state and $S_{i+1} = S_i + \Delta_r(S_i)$ holds for all $i \geq 0$. If for some k we have $S_{k+1} = S_k$ (and then further $S_i = S_k$ for all $i \geq k$), we call S_k a *final* state and say that the run *terminates* in S_k .

3.2 Nesting of RDF Data and SPARQL Queries

In ASMs RDF data can be simply represented by a single ternary function symbol \mathcal{R} . In a state S we always have $\mathcal{R}(s, p, o) \in \{\text{true}, \text{false}\}$ with the obvious interpretation that $\mathcal{R}(s, p, o) = \text{true}$ holds iff the triplet (s, p, o) belongs to the RDF data \mathcal{R} . As common in databases we tacitly assume that in every state almost all triplets have the value *false*.

In the presence of meta-data stratification can be checked by the following ASM rule, which constructs an assignment of strata, if possible. For convenience we also assign a stratum number 0 to all labels not in \mathcal{St} . The main idea is that in every step (in the ‘progress’ mode) the algorithm assigns an stratum number to every label ℓ in \mathcal{St} which has not got one yet and which represents a triplet (s, p, o) where the stratum numbers of s and o have already been defined. This process continues until it is no longer possible to assign new stratum numbers. If at this point all labels in \mathcal{St} have got a stratum number, then the data has been successfully stratified.

```

STRATIFIED  $\equiv$ 
  PARBLOCK
  IF   mode = ‘init’
  THEN PARBLOCK
    FORALL  $\ell \in \mathcal{I} \cup \mathcal{A} \cup \mathcal{V}$  WITH  $\neg \exists (s, p, o) \in \mathcal{R}. \text{represents}(\ell, (s, p, o))$ 
    DO strat( $\ell$ ) := 0 ENDDO
    mode := ‘progress’

```

```

    ENDPAR    ENDIF
IF    mode = 'progress'
THEN  FORALL  $\ell \in \mathcal{A}$ 
      WITH strat( $\ell$ ) = undef  $\wedge \exists(s, p, o) \in \mathcal{R}. \text{represents}(\ell, (s, p, o)) \wedge$ 
        strat( $s$ )  $\neq$  undef  $\wedge$  strat( $o$ )  $\neq$  undef
      DO    CHOOSE  $(s, p, o) \in \mathcal{R}$  WITH represents( $\ell, (s, p, o)$ )
          DO strat( $\ell$ ) := max(strat( $s$ ), strat( $o$ )) + 1 ENDDO
      ENDDO    ENDIF
IF    mode = 'progress'  $\wedge \forall \ell \in \mathcal{A}. \text{strat}(\ell) = \text{undef} \wedge \exists(s, p, o) \in \mathcal{R}. \text{represents}(\ell, (s, p, o)) \Rightarrow$ 
      strat( $s$ ) = undef  $\vee$  strat( $o$ ) = undef
THEN  mode := 'limit'    ENDIF
IF    mode = 'limit'
THEN  IF  $\exists \ell \in \mathcal{A}. \text{strat}(\ell) = \text{undef}$ 
      THEN stratified := false
      ELSE stratified := true
      ENDF    ENDIF    ENDPAR

```

The following ASM rule transforms given RDF data with meta-data into nested RDF data, provided the original data are stratified. For all auxiliary label $_s$ and all triplet (s_1, p_1, o_1) represented by $_s$ where neither s_1 nor o_1 are auxiliary labels which represent triplets, the algorithm replaces every occurrence of $_s$ in every triplet in \mathcal{R} by the actual triplet (s_1, p_1, o_1) . This process is repeated until no more replacements are possible.

NEST \equiv

```

FORALL  $\_s, s_1, p_1, o_1$ 
WITH  $\_s \in \mathcal{A} \wedge \mathcal{R}(s_1, p_1, o_1) \wedge \text{represents}(\_s, (s_1, p_1, o_1)) \wedge$ 
   $\neg \exists s_2, p_2, o_2. \text{represents}(s_1, (s_2, p_2, o_2)) \wedge$ 
   $\neg \exists s_3, p_3, o_3. \text{represents}(o_1, (s_3, p_3, o_3))$ 
DO  PARBLOCK
  FORALL  $s, p, o$  WITH  $\mathcal{R}(s, p, o) \wedge (s = \_s \vee o = \_s)$ 
  DO  PARBLOCK
    IF  $s = \_s \wedge o = \_s$ 
    THEN PARBLOCK
       $\mathcal{R}((s_1, p_1, o_1), p, (s_1, p_1, o_1)) := \text{true}$ 
       $\mathcal{R}(\_s, p, \_s) := \text{false}$ 
    ENDPAR    ENDIF
    IF  $s = \_s \wedge o \neq \_s$ 
    THEN PARBLOCK
       $\mathcal{R}((s_1, p_1, o_1), p, o) := \text{true}$ 
       $\mathcal{R}(\_s, p, o) := \text{false}$ 
    ENDPAR    ENDIF
    IF  $s \neq \_s \wedge o = \_s$ 
    THEN PARBLOCK
       $\mathcal{R}(s, p, (s_1, p_1, o_1)) := \text{true}$ 

```

```

         $\mathcal{R}(s, p, \_s) := false$ 
      ENDPAR    ENDIF
    ENDDO
     $\mathcal{R}(\_s, :subject, s_1) := false$ 
     $\mathcal{R}(\_s, :predicate, p_1) := false$ 
     $\mathcal{R}(\_s, :object, o_1) := false$ 
  ENDPAR
ENDDO

```

The following ASM rule transforms a given query into a nested one, provided the original RDF data are stratified. Note that only the RDF patterns in \mathcal{P} are changed, whereas the result variables remain unchanged. For all auxiliary label $_s$ and all triplet (s_1, p_1, o_1) represented by $_s$ where neither s_1 nor o_1 are auxiliary labels which represent triplets, the algorithm replaces every occurrence of $_s$ in every triplet in \mathcal{P} by the actual triplet (s_1, p_1, o_1) . For the case of variables, we say that $\$s$ *q-represents* a triplet (s, p, o) if $(\$s, :subject, s)$, $(\$s, :predicate, p)$ and $(\$s, :object, o)$ all three belong to \mathcal{P} . As with labels, for every variable $\$s$ in V and all triplet (s_1, p_1, o_1) *q-represented* by $\$s$ where neither s_1 nor o_1 *q-represent* triplets, the algorithm replaces every occurrence of $\$s$ in every triplet in \mathcal{P} by the actual triplet (s_1, p_1, o_1) . Again these processes are repeated until no more replacements are possible.

```

NEST-QUERY  $\equiv$ 
  PARBLOCK
    FORALL  $\_s, s_1, p_1, o_1$ 
    WITH  $\_s \in \mathcal{A} \wedge \mathcal{R}(s_1, p_1, o_1) \wedge \text{represents}(\_s, (s_1, p_1, o_1)) \wedge$ 
       $\neg \exists s_2, p_2, o_2. \text{represents}(s_1, (s_2, p_2, o_2)) \wedge$ 
       $\neg \exists s_3, p_3, o_3. \text{represents}(o_1, (s_3, p_3, o_3))$ 
    DO    FORALL  $s, p, o$  WITH  $(s, p, o) \in \mathcal{P} \wedge (s = \_s \vee o = \_s)$ 
      DO    PARBLOCK
        IF  $s = \_s \wedge o = \_s$ 
        THEN  $\mathcal{P} := \mathcal{P} \cup \{((s_1, p_1, o_1), p, (s_1, p_1, o_1))\}$ 
           $-\{(\_s, p, \_s), (\_s, :subject, s_1),$ 
             $(\_s, :predicate, p_1), (\_s, :object, o_1)\}$ 
        ENDIF
        IF  $s = \_s \wedge o \neq \_s$ 
        THEN  $\mathcal{P} := \mathcal{P} \cup \{((s_1, p_1, o_1), p, o)\}$ 
           $-\{(\_s, p, o), (\_s, :subject, s_1),$ 
             $(\_s, :predicate, p_1), (\_s, :object, o_1)\}$ 
        ENDIF
        IF  $s \neq \_s \wedge o = \_s$ 
        THEN  $\mathcal{P} := \mathcal{P} \cup \{(s, p, (s_1, p_1, o_1))\}$ 
           $-\{(s, p, \_s), (\_s, :subject, s_1),$ 
             $(\_s, :predicate, p_1), (\_s, :object, o_1)\}$ 
        ENDIF
      ENDPAR
    ENDPAR
  ENDPAR

```



```

        ENDDO
    ENDDO
    FORALL $s, s_1, p_1, o_1
    WITH $s \in V \wedge (s_1, p_1, o_1) \in \mathcal{P} \wedge \text{q-represents}(\$s, (s_1, p_1, o_1)) \wedge
        \neg \exists s_2, p_2, o_2. \text{q-represents}(s_1, (s_2, p_2, o_2)) \wedge
        \neg \exists s_3, p_3, o_3. \text{q-represents}(o_1, (s_3, p_3, o_3))
    DO    FORALL s, p, o WITH (s, p, o) \in \mathcal{P} \wedge (s = \$s \vee o = \$s)
        DO    PARBLOCK
            IF s = $s \wedge o = $s
            THEN \mathcal{P} := \mathcal{P} \cup \{((s_1, p_1, o_1), p, (s_1, p_1, o_1))\}
                - \{(\$s, p, \_s), (\$s, :subject, s_1),
                    (\$s, :predicate, p_1), (\$s, :object, o_1)\}
            ENDF
            IF s = $s \wedge o \neq $s
            THEN \mathcal{P} := \mathcal{P} \cup \{((s_1, p_1, o_1), p, o)\}
                - \{(\$s, p, o), (\$s, :subject, s_1),
                    (\$s, :predicate, p_1), (\$s, :object, o_1)\}
            ENDF
            IF s \neq $s \wedge o = $s
            THEN \mathcal{P} := \mathcal{P} \cup \{(s, p, (s_1, p_1, o_1))\}
                - \{(s, p, \$s), (\_s, :subject, s_1),
                    (\_s, :predicate, p_1), (\$s, :object, o_1)\}
            ENDF
        ENDPAR
    ENDDO
ENDDO
ENDPAR

```

3.3 Query Evaluation

The evaluation of a query Q can then be easily expressed by the following ASM rule:

```

EVALUATE \equiv  FORALL  $x_1, \dots, x_n \in \mathcal{I} \cup \mathcal{A} \cup \mathcal{V}$ 
                WITH  $\forall (s, p, o) \in \mathcal{P}. \mathcal{R}(s, p, o)$ 
                DO
                    Result( $x_{i_1}, \dots, x_{i_m}$ ) := true
                ENDDO

```

using the assumption $V_{\mathcal{P}} = \{x_1, \dots, x_n\}$ and $V_q = \{x_{i_1}, \dots, x_{i_m}\}$.

Note that this rule is applicable regardless if we have nested RDF data and a nested SPARQL query or not. Then we can use the following main ASM rule with the initial value ‘init’ of ctl-state to express the complete evaluation procedure:

```

MAIN \equiv  PARBLOCK
            IF ctl-state = ‘init’ \wedge stratified = undef

```

```

THEN STRATIFIED
ENDIF
IF ctl-state = 'init' ∧ stratified = false
THEN ctl-state := 'eval'
ENDIF
IF ctl-state = 'init' ∧ stratified = true
THEN ctl-state := 'nest'
ENDIF
IF   ctl-state = 'nest' ∧ stratified = true
THEN PARBLOCK
      NEST
      NEST-QUERY
      ctl-state := 'eval'
    ENDPAR
ENDIF
IF   ctl-state = 'eval'
THEN PARBLOCK
      EVALUATE
      ctl-state := 'halt'
    ENDPAR
ENDIF
ENDPAR

```

4 Correctness and Complexity Analysis

In this section we analyse the evaluation procedure, which we specified in the previous section, with respect to correctness and complexity. For correctness we emphasise that the result of a nested query on nested RDF data preserves the result that would have been obtained without the nesting. For complexity we show that by nesting we can expect a reduction of the number of necessary matches between patterns and RDF data.

4.1 Correctness of the Abstract State Machine

We first look at the rule STRATIFIED, for which we show termination and correctness.

Proposition 4.1. *The ASM rule STRATIFIED always reaches a final state, in which no more changes are possible. In this final state the value at the location (stratified, ()) is either true or false.*

Proof. Initially we have mode = 'init', so in a first step we will assign $\text{strat}(\ell) := 0$ to all labels $\ell \in \mathcal{I} \cup \mathcal{V}$ and as well to all labels $\ell \in \mathcal{A}$ that do not represent statements. Furthermore, the value of location (mode, ()) will become 'progress'.

In a state with mode = ‘progress’ either at least one label $\ell \in \mathcal{A}$, for which $\text{strat}(\ell)$ is still undefined, will be assigned a stratum number—and this cannot be repeated forever, as there are only finitely many such labels—or if this is not possible, i.e. the condition in the second branching rule is violated, then the value of location (mode, ()) will become ‘limit’.

In a state with mode = ‘limit’ only a value *true* or *false* is assigned to the location (stratified, ()), and then no further changes are possible. \square

Proposition 4.2. *The ASM rule STRATIFIED produces the result stratified = true iff the RDF data \mathcal{R} are stratified in the sense of Definition 2.3.*

Proof. According to Proposition 4.1 the rule STRATIFIED produces either the result stratified = *true* or stratified = *false*. In the former case $\text{strat}(\ell)$ is defined for all labels. Labels for $\ell \in \mathcal{St}$ have been assigned in a step on a state, in which mode = ‘progress’ holds. The assignment is done such that $\text{strat}(\ell) > \max(\text{strat}(s), \text{strat}(o))$ holds, when ℓ represents an RDF triplet (s, p, o) , which is exactly the condition required in Definition 2.3.

Conversely, in case stratified = *false* results, there exists a label $\ell \in \mathcal{St}$ such that $\text{strat}(\ell)$ remains undef in the final state. According to the condition in the second branching rule (and its negation in the third one) ℓ represents an RDF triplet (s, p, o) , in which for at least one of s or o (let this be ℓ_1) $\text{strat}(\ell_1)$ remains also undef in the final state. So we obtain a sequence $\ell = \ell_0, \ell_1, \ell_2, \dots$ of labels in \mathcal{St} , for which $\text{strat}(\ell_i) = \text{undef}$ holds for all i . As \mathcal{St} is finite, this sequence must contain a cycle, say $\ell_k = \ell_{k'}$ (with $k < k'$). Then it is impossible to assign stratum number to labels ℓ_i with $k \leq i \leq k'$ fulfilling the condition in Definition 2.3, i.e. the RDF data are not stratified. \square

For correctness of the ASM rule MAIN we first show the termination of the nesting rules.

Proposition 4.3. *If RDF data \mathcal{R} with meta-data are stratified, then the ASM rules NEST and NEST-QUERY for a query (V_q, \mathcal{P}) always reach a final state, in which no more changes are possible.*

Proof. Though stratum numbers are not explicitly used in the rule NEST, the nested proceeds according to increasing stratum numbers. If $\ell \in \mathcal{St}$ has stratum number $\text{strat}(\ell) = 1$ and represents (s, p, o) , then s and o must have stratum number 0. Then any occurrence of ℓ in an RDF triplet as subject or object will be replaced by (s, p, o) and the corresponding meta-data will be deleted. As this is done in parallel for all such labels $\ell \in \mathcal{St}$, any remaining labels in \mathcal{St} , i.e. representing some RDF triplets, will have a larger stratum number. That is, if the lowest stratum number left is i , then for $\text{strat}(\ell) = i$, if ℓ represents (s, p, o) , then s and o do not represent RDF triplets, so the nesting, i.e. the replacement of occurrences, will succeed. This continues to the maximum stratum number, when no more change is possible, so NEST terminates.

The very same procedure is applied for the nesting of a query (V_q, \mathcal{P}) by the rule NEST-QUERY. So, using the same argument we conclude that NEST-QUERY terminates in a state, where no further change is possible. \square

Finally we show that the rule MAIN is correct, i.e. nesting of data and queries in the case of stratified RDF data and meta-data with follow-on evaluation by the rule EVALUATE preserves the result of the queries.

Proposition 4.4. *Let (V_q, \mathcal{P}) be a query on RDF data \mathcal{R} with meta-data. Then the result of the ASM rule MAIN satisfies the required condition (2.1).*

Proof. The termination of MAIN is an obvious consequence of Proposition 4.3 and the fact that EVALUATE only makes a single step. If the RDF data are not stratified, then there is no change to the evaluation, so we can assume without loss of generality that \mathcal{R} is stratified.

Furthermore, using induction it suffices to consider a single replacement step in NEST and NEST-QUERY. That is, if we have a pattern $(\$s, p, o) \in \mathcal{P}$ and this matches an RDF triplet $(_s, p', o')$, for which $\text{represents}(_s, (s_1, p_1, o_1))$ holds, then we have RDF triplets $(_s, \text{:subject}, s_1)$, $(_s, \text{:predicate}, p_1)$ and $(_s, \text{:object}, o_1)$. Through NEST these meta-data will be replaced by the nested RDF triplet $((s_1, p_1, o_1), p', o')$.

The query may then contain patterns $(\$s, \text{:subject}, s'_1)$, $(\$s, \text{:predicate}, p'_1)$ and $(\$s, \text{:object}, o'_1)$, for which $\text{q-represents}(\$s, (s'_1, p'_1, o'_1))$ holds. These match the RDF triplets for the representation of meta-data. Through NEST-QUERY these patterns will be replaced by the nested pattern $((s'_1, p'_1, o'_1), p, o)$. This nested pattern matches the nested RDF triplet iff the original patterns match the original RDF triplets, and consequently the nesting of RDF data and query patterns preserves the bindings for the query result.

The same argument applies analogously for variables in the third argument, and in the same way for constants, which implies the preservation of query results under nesting of RDF data and query patterns. \square

4.2 Reduction of Complexity

When looking at the complexity of query evaluation, the decisive condition in EVALUATE is $\forall (s, p, o) \in \mathcal{P}. \mathcal{R}(s, p, o)$, i.e. we have to consider all bindings of variables in patterns (s, p, o) that will simultaneously define RDF triplets in \mathcal{R} . We may assume that the time required to check, if an instantiation of a pattern matches an RDF triplet in the database is bound by a constant. While patterns are interpreted conjunctively, we still have to match all patterns against the given RDF data, which implies that the time complexity for the evaluation of a query depends linearly on $N \cdot P$, where N is the number of RDF triplets in the database and P is the number of patterns in the query.

So, let n_0 be the number of RDF triplets in \mathcal{R} , and let $n'_0 \leq n_0$ denote the number of statements, about which meta-data is stored. Then we get $3 \cdot n'_0 + n_1$ adding RDF triplets for the meta-data, where n_1 is the number of additional RDF triplets containing the relevant meta-data, while $3 \cdot n'_0$ additional RDF triplets are required for linking the meta-data to the core data. We can assume $n'_0 \leq n_1$, as it does not make sense to introduce auxiliary labels for the representation of RDF triplets, if they do not appear in at least one meta RDF triplet.

In the same way we can consider meta-meta-data, etc. so we get $n_0, n_1, n_2, \dots, n_k$ for the numbers of additional RDF triplets as well as $n'_0, n'_1, n'_2, \dots, n'_{k-1}$ for the numbers of those triplets that are represented by meta-data. As before we have $n'_i \leq n_i$ and $n'_i \leq n_{i+1}$. The total number of RDF triplets amounts to

$$N = \sum_{i=0}^k n_i + 3 \cdot \sum_{i=0}^{k-1} n'_i,$$

while the number of nested RDF triplets after the running of NEST is only $N_{nest} = \sum_{i=0}^k n_i$. Similarly, we obtain $P = \sum_{i=0}^k p_i + 3 \cdot \sum_{i=0}^{k-1} p'_i$ for the number of patterns and $P' = \sum_{i=0}^k p_i$ for the number of nested patterns after running NEST-QUERY. As the number of patterns in queries is usually not very high, the reduction from P to P' does not contribute very much to the reduction of complexity. If we let $P' \approx P$, then the number of matches is still reduced significantly by $3 \cdot P \cdot \sum_{i=0}^{k-1} n'_i$.

Clearly, any optimisation of the way RDF triplets are stored impacts on the complexity of query evaluation, but this will be similar for unnested or nested triplets.

5 Concluding Remarks

In this paper we investigated the evaluation of SPARQL queries on RDF data with meta-data. Using specifications by Abstract State Machines we showed that a reduction of complexity can be obtained, if RDF data and consequently also patterns in SPARQL queries are nested. This reduction permits RDF statements to be used directly within other RDF statements and thus reduces the number of matches in the evaluation of queries. This nesting is only possible, if the RDF data and meta-data are stratified in the sense that there are no cycles in the meta-data. Stratification can also be checked by a simple ASM.

In doing so we provided a solid formal underpinning for the idea of nesting that was brought up by Hartig [8]. In particular, we provided proofs that the evaluation of nested queries on nested RDF data preserves the results that would have been obtained without the nesting. By introducing stratification we also simplified the condition, under which nesting is possible. We also showed that there is indeed a reduction of complexity.

From a practical perspective, the procedure presented in this paper could be effectively implemented in parallel using some semantic Web tools, for instance, following the approach described in [6].

References

1. Benczúr, A., Hajas, C., Kovács, G.: Datalog extension for nested relations. *Comput. Math. Appl.* **30**(12), 51–79 (1995)
2. Börger, E., Stärk, R.F.: *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer, Heidelberg (2003). <https://doi.org/10.1007/978-3-642-18216-7>

3. Conen, W., Klapsing, R., Köppen, E.: RDF M&S revisited: from reification to nesting, from containers to lists, from dialect to pure XML. In: Cruz, I.F., et al. (eds.) *The Emerging Semantic Web, Selected Papers from the First Semantic Web Working Symposium*, Stanford University, California, USA, 30 July–1 August 2001. *Frontiers in Artificial Intelligence and Applications*, vol. 75. IOS press (2002)
4. Cyganiak, R., Wood, D., Lanthaler, M.: *RDF 1.1 Concepts and Abstract Syntax*, W3C Recommendation, 25 February 2014 (2014). <https://www.w3.org/TR/rdf11-concepts/>
5. Geerts, F., Karvounarakis, G., Christophides, V., Fundulaki, I.: Algebraic structures for capturing the provenance of SPARQL queries. In: Tan, W.C., et al. (eds.) *Joint 2013 EDBT/ICDT Conferences (ICDT 2013)*, pp. 153–164. ACM (2013)
6. Gombos, G., Kiss, A.: P-Spar(k)ql: SPARQL evaluation method on Spark GraphX with parallel query plan. In: 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud), pp. 212–219 (2017)
7. Hartig, O.: *Querying a Web of Linked Data - Foundations and Query Execution*. *Studies on the Semantic Web*, vol. 24. IOS Press, Amsterdam (2016)
8. Hartig, O.: Foundations of RDF* and SPARQL* - an alternative approach to statement-level metadata in RDF. In: Reutter, J.L., Srivastava, D. (eds.) *Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web*. *CEUR Workshop Proceedings*, vol. 1912. CEUR-WS.org (2017)
9. Hayes, P.J., Patel-Schneider, P.F.: *RDF 1.1 Semantics*, W3C recommendation, 25 February 2014 (2014). <https://www.w3.org/TR/rdf11-mt/>
10. Hernández, D., Hogan, A., Krötzsch, M.: Reifying RDF: what works well with wiki-data? In: Liebig, T., Fokoue, A. (eds.) *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems (ISWC 2015)*. *CEUR Workshop Proceedings*, vol. 1457, pp. 32–47. CEUR-WS.org (2015)
11. Nguyen, V., Bodenreider, O., Sheth, A.P.: Don't like RDF reification?: making statements about statements using singleton property. In: Chung, C.W., et al. (eds.) *23rd International World Wide Web Conference (WWW 2014)*, pp. 759–770. ACM (2014)
12. Pérez, J., Arenas, M., Gutiérrez, C.: Semantics and complexity of SPARQL. *ACM Trans. Database Syst.* **34**(3), 16:1–16:45 (2009)
13. Pham, M.-D., Boncz, P.: Exploiting emergent schemas to make RDF systems more efficient. In: Groth, P., et al. (eds.) *ISWC 2016*. LNCS, vol. 9981, pp. 463–479. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46523-4_28
14. Tsialiamanis, P., Sidiropoulos, L., Fundulaki, I., Christophides, V., Boncz, P.A.: Heuristics-based query optimisation for SPARQL. In: Rundensteiner, E.A., et al. (eds.) *15th International Conference on Extending Database Technology (EDBT 2012)*, pp. 324–335. ACM (2012)



SIMD Vectorized Hashing for Grouped Aggregation

Bala Gurumurthy^(✉), David Broneske, Marcus Pinnecke, Gabriel Campero, and Gunter Saake

Otto-von-Guericke-Universität, Magdeburg, Germany
{bala.gurumurthy,david.broneske,marcus.pinnecke,gabriel.campero,
gunter.saake}@ovgu.de

Abstract. Grouped aggregation is a commonly used analytical function. The common implementation of the function using hashing techniques suffers lower throughput rate due to the collision of the insert keys in the hashing techniques. During collision, the underlying technique searches for an alternative location to insert keys. Searching an alternative location increases the processing time for an individual key thereby degrading the overall throughput. In this work, we use Single Instruction Multiple Data (SIMD) vectorization to search multiple slots at an instant followed by direct aggregation of results. We provide our experimental results of our vectorized grouped aggregation with various open-addressing hashing techniques using several dataset distributions and our inferences on them. Among our findings, we observe different impacts of vectorization on these techniques. Namely, linear probing and two-choice hashing improve their performance with vectorization, whereas cuckoo and hopscotch hashing show a negative impact. Overall, we provide in this work a basic structure of a dedicated SIMD accelerated grouped aggregation framework that can be adapted with different hashing techniques.

Keywords: SIMD · Hashing techniques · Hash based grouping
Grouped aggregation · Direct aggregation · Open addressing

1 Introduction

Many analytical processing queries (e.g., OLAP) are directly related to the efficiency of their underlying functions. Some of these internal functions are time-consuming and even require the complete dataset to be processed before producing the results. One of such compute-intensive internal functions is a grouped aggregation function that affects the overall throughput of the user query. Hence, it is evident that improving the throughput of grouped aggregation processing in-turn improves the overall query throughput.

A grouped aggregation function is commonly implemented using hashing techniques. The throughput of a grouped aggregation function is affected by the

collision of keys in the underlying hashing techniques. A collision occurs when a given key is hashed to a slot that is preoccupied with another key. In this case, the collision is resolved by finding an alternative location for the colliding key. Searching for an alternative location requires more time and affects the throughput of the whole operation. Hence, improving probe time in a hashing technique improves the overall execution time.

Since real-time systems process large volumes of data, Single Instruction Multiple Data (SIMD) is the commonly sought out parallelization strategy to perform these operations. Previous work has shown how to use SIMD to accelerate different database processing operations [3, 7, 11]. Ross et al., detail a strategy of using SIMD acceleration for probing multiple slots in cuckoo hashing [14]. Furthermore, Broneske et al. have shown that execution of hand-written database operations that are code optimized for underlying hardware can be faster than the query plan given by a query optimizer [1, 2]. Using these insights, we explore the advantages of hand-written grouped-aggregation using SIMD acceleration over various hashing techniques. In addition to it, we also use the code optimization strategy of direct aggregation for improved throughput (explained in Sect. 3).

We discuss a SIMD-based probing called horizontal vectorization, where multiple slots are searched for a single key at an instant using the instructions available in the SIMD instruction set [10]. We adopt this technique to different open-addressing hashing techniques available. This vectorized probing is enhanced with direct aggregation for increased performance. From our evaluation results, we observe that these optimizations provide notable speedups of up to N x over the scalar mechanism, where N is the vector length. Our core contributions in the paper are:

- We explore SIMD-based code optimization for different hashing techniques (Sect. 3).
- We evaluate the performance of computing grouped aggregation using these SIMD accelerated techniques (Sect. 5.2).
- Finally, we report our observations on the results and discuss the impact of SIMD for the different hashing techniques (Fig. 10).

From our evaluation, we found out that the multiple parameters involved in hashing techniques also affect their efficiency. Hence, SIMD accelerated grouped aggregation could be further improved by tuning these underlying hashing technique’s related parameters. Also, based on our results, we show that the best hashing algorithm for a given data distribution can be selected for reduced latency.

2 Related Work

In this chapter, we compare our work with others that are similar in studying the aggregation operation in databases. The approaches are selected based on the use of SIMD and other hardware related optimizations over grouped aggregation functions.

In this work, we do not consider quadratic probing and double hashing. We exclude them due to their poor locality of keys.

Jiang et al. use a modified bucket chaining hashing technique to group data and aggregate them on the fly during insertion [7]. In order to eliminate conflicts arising due to SIMD parallelization with insertion and to accommodate values within main memory, they have proposed to add a distinctive offset for each of the SIMD lanes and manipulate data separately. This approach is also extended for MIMD execution. Broneske et al. use SIMD to accelerate selection operations [1]. The authors argue that SIMD acceleration influences the operation of aggregation operations. In our approach, we have gained similar improvements in SIMD acceleration of grouping with aggregation operation especially for linear probing and two choice hashing. Further, various SIMD-based aggregation operations are explained in the paper by Zhou and Ross [15]. They detail the performance impact of SIMD on different aggregation operations.

Finally, Richter et al. provided an extensive analysis of different hashing techniques [13]. These are some of the works done on hashing and grouped aggregation techniques. In the next section, we discuss different hashing techniques followed by details on incorporating SIMD on the them.

3 Hash-Based Grouped Aggregation

Traditional grouped aggregation using hashing techniques is computed in two phases. In the first phase, all the keys are hashed and segregated into their own buckets. In the second phase, each of the keys within the buckets is aggregated providing the final result. Since it is time-consuming to perform two passes on the same data, an alternative single pass variant - *direct aggregation* is used to improve the throughput.

Direct aggregation initializes the aggregate value for a key during its insertion. During subsequent insertions of the same key, its corresponding aggregate value is updated. To further improve the throughput of the single pass algorithm, SIMD vectorization can be used.

In this section, we describe the different hashing techniques used for computing grouped aggregation followed by our strategy to incorporate SIMD vectorization on these techniques.

3.1 Outline of Hashing Techniques

The major bottleneck of any hashing technique is the collision of keys, where two keys are hashed into the same location. Each of the hashing techniques has their own ways to resolve the collision of keys.

We use the open-addressing hashing techniques: linear probing, two-choice hashing, hopscotch hashing and cuckoo hashing for computing grouped aggregation. These techniques have a linear hash-table structure with a known boundary which is suitable for vectorization. In the following sections, we detail the collision resolution mechanism of these techniques and the ways to perform grouped aggregation on them.

3.2 Cuckoo Hashing

Cuckoo hashing resolves collision by using multiple hash tables [9]. These tables resolve collision by swapping collided keys. The collided key, during collision, replaces the key within the insert table. The replaced key is then hashed into the next hash table. The keys are swapped until the final table. In case of collision in final table, the replaced key is routed back to the first table and swapping continues. Since number of swaps is in-determinant, a user-defined threshold, *swap number* cuts-off, swapping and re-hash the whole keys using different hashing functions for the tables and increasing the hash table size. Hence, cuckoo hashing has near direct probing by trading-off increased insertion time. For grouped aggregation computation, first the given key is probed in all the locations in the hash tables. If it is found, the corresponding payload is updated. Else, the key along with the initial payload are inserted. Both the key and payload are swapped among the tables for accommodating the new key.

3.3 Linear Probing

Linear probing, as the name suggests, searches the hash table for a desired location using sequential scanning [4]. It searches the hash table to find the nearest empty slot for insertion. Similarly, it scans the table linearly for the probe value. For grouped aggregation, we search the hash table until either the probe key or an empty location is found. If an empty slot is encountered first, then aggregate resultant is initialized. Else if the key is found, the aggregate is updated.

3.4 Two-Choice Hashing

Two-choice hashing, a variant of linear probing, has two different hash functions for a single hash table [12]. The two hash functions provide for a given key, two alternative positions. Having alternative positions increase the chance of finding a slot with lesser probe time. If both of the slots are occupied, the hash table is probed linearly from both the slots until an empty slot is encountered. Grouped aggregation is computed similar to linear probing. However, instead of a single probe, the hash table is probed from the two start locations.

3.5 Hopscotch Hashing

Hopscotch hashing has a user-defined parameter - neighborhood - that guarantees any given key will be available within the neighborhood range from the originally hashed location [6]. If the key is not available within the neighborhood, it has to be inserted. For insertion of a key, the hash table is searched for an empty location and the existing keys in the table are swapped until an empty slot is available within the neighborhood. Finally, the key is inserted in the slot. Thus, hopscotch hashing trades less search space to prolonged insertion.

4 SIMD Vectorization of Hashing Techniques

SIMD performs a single instruction on multiple data in an instant. Modern CPUs have extended register sizes of up to 128 bits, possible for accommodating four packed integers. Using these larger registers, a single operation is applied over multiple data at a time. Though it might seem trivial that K packed values provide a speed-up of K in SIMD, incorporating SIMD in hashing techniques has its complexities due to the collision of keys.

Specifically, a hashing technique has delayed time due to the search of the desired location either for inserting a new key or searching an existing key. We address this issue by incorporating SIMD for searching multiple slots at a time thereby improving the throughput. In the next sections, we detail the SIMD accelerated probing on the above discussed hashing techniques.

4.1 Table Structure

Probing requires multiple slots of a given hash table to be readily available. Hence, a right table structure improves the efficiency of the hashing technique. Since cuckoo hashing has multiple hash tables, we use the table structure described in Fig. 2. We store a packed set of keys followed by a packed set of payloads for each bucket. As multiple swaps are required, packing keys and payloads improves the efficiency by loading both into the memory for easy swapping among tables.

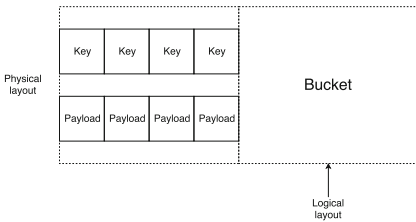


Fig. 1. SoA table structure

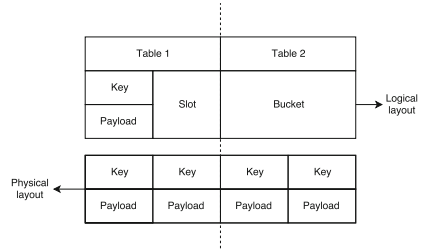


Fig. 2. Cuckoo hash-table structure

For other hashing techniques, we use a Structure of Arrays (SoA) for the hash table as shown in Fig. 1. The hash tables have keys and payloads in different arrays with the same index pointing to a key and its corresponding index. We use this structure, as the payload is accessed only if the key is found in the hash table.

4.2 SIMD Accelerated Cuckoo Hashing

Ross et al., have given a detailed outline for performing SIMD accelerated probing in cuckoo hashing [14]. We extend their idea by adding the direct aggregation

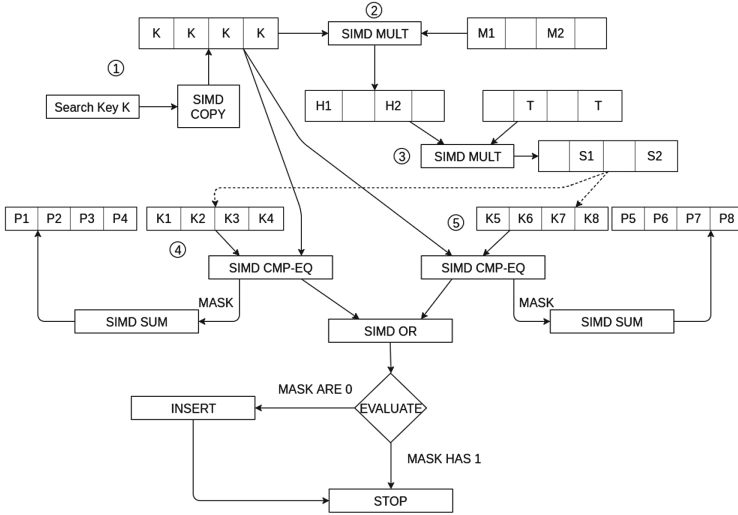


Fig. 3. SIMD accelerated cuckoo hashing (extended from Ross et al. [14])

mechanism. We depict the general direct aggregation with probing in cuckoo hashing in Fig. 3.

Grouped aggregation using cuckoo hashing has two phases. First, the slot locations are identified using hashing functions. Second, we use the identified slots to probe the respective tables.

In our approach, the slots for a key are identified using multiplicative hashing. The function multiplies the key with a random multiplier, then performs modulo on the resultant with the table size to get the slots. To perform multiplicative hashing in SIMD, a SIMD COPY functions (e.g., `_mm_set_epi32()`) vectorize the search key. This key vector is multiplied with a multiplier vector using SIMD MULT functions (e.g., `_mm_mul_epi32()`) (2). Finally, the resultant vector is again multiplied with a table size vector (3). The slot values are available in the least significant bytes of the result vector.

Based on the slots, the bucket vector values are compared with the key vector using SIMD CMP-EQ functions (e.g., `_mm_cmpeq_epi32()`) providing a mask vector (4, 5). The mask vector based on the comparison has either 0 or 1 and this result along with the corresponding payloads are updated using SIMD SUM operations (e.g., `_mm_add_epi32()`). Finally, if all the masks are 0 then the key is inserted.

4.3 SIMD-Accelerated Linear Probing

Scanning hash table one at a time is time consuming. This is improved by using SIMD for scanning multiple slots in an instant.

In Fig. 4, we describe the SIMD adapted linear probing mechanism. First, scalar multiplicative hashing function computes the slot for the given key. We use

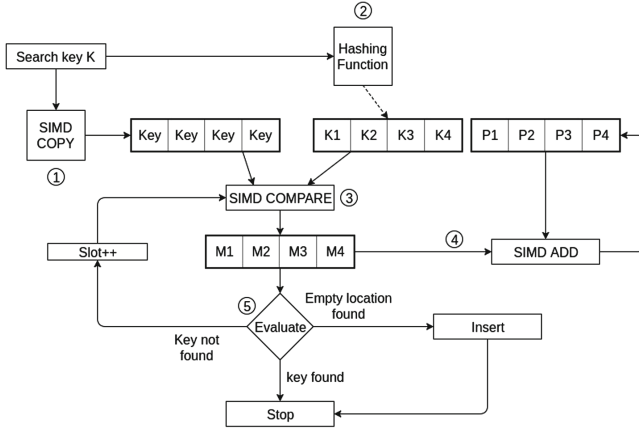


Fig. 4. SIMD accelerated linear probing

scalar function as only one slot is required. Second, the search key is vectorized using a SIMD COPY function (1). This search key vector is then compared with the values present in the pointed bucket using SIMD COMPARE (e.g., `_mm_cmpeq_epi32()`). Final steps are similar to that of cuckoo hashing, where the resultant mask vector is added to the corresponding payloads (4) and insertion based on the mask is determined (5).

4.4 SIMD Accelerated Two-Choice Hashing

The major advantage of using SIMD for two-choice hashing is to compute the slots from two hash functions at a time. SIMD acceleration is similar to that of linear probing. Instead of comparing the key in a single vector of slots, we compare the vectors from the selected slots in sequence.

4.5 SIMD Accelerated Hopscotch Hashing

Hopscotch hashing is a multi-step algorithm, where the probing for key is done in the first and swapping of keys to have empty location in the second. Hence, this technique requires additional measures for adapting SIMD. In fact, it might even create an overhead for preprocessing the input to adapt SIMD for each of these steps. We call these steps forward and reverse probe. In the forward probe, we search for a key until the neighborhood boundary and empty space afterward. In reverse probe, we perform the swaps with the empty slots until we reach the neighborhood. We use SIMD to accelerate the probing of keys in the forward probe and swapping of keys while insertion.

We use the same SIMD acceleration of linear probing for the forward probe in hopscotch hashing (cf. Fig. 4). For swapping of keys, we use the gather instructions in SIMD as given in Fig. 5.

Forward Probe. Within the neighborhood, direct aggregation is done similar to linear probing. Once outside the boundary, the table is probed for an empty location. This empty location index is used in the next phase for swapping of keys.

Reverse Probe. A reverse probe is done to select the positions to swap the keys in order to have an empty location within the neighborhood boundary. During the reverse probe, the pointer moves back from the empty slot until a key inside the neighborhood of insert key can be swapped. On each step, the key to be swapped is stored into a swap array until the neighborhood of the insert key is reached. The keys are swapped for the indexes and the given key is inserted. If the neighborhood is not reached, then the table has to be rehashed.

In the next section, we evaluate these SIMD-accelerated hashing techniques using different data distributions. We detail our evaluation setup first, followed by our results. Finally, we provide our insights on the results found.

5 Evaluation

To assess the efficiency of the presented hashing techniques, we measure their performance for different data distributions. In our hashing techniques, the insertion and aggregation functions are performed in a single atomic step. Hence, we include the overall insertion and probing time for determining the efficiency of the techniques. We conducted our experiments on a machine running CentOS Linux version-7.1.1503 and gcc version 4.8.5 with an octa core Intel Xeon E5-2630 v3s- 2014. The system is incorporated with the AVX2 instruction set.

5.1 General Assumptions and Experimental Setup

For all our experiments we have the below assumptions for terms of consistency. These are common across all the hashing techniques.

Key-payload pair: We assume that the key and payload to insert are 32-bit integers. Also, w.l.o.g., the value zero is not a valid key or payload value as it is used to represent empty slots in the hash table. In a production system, the maximum value in the domain can also be chosen.

Hash function: A multiplicative hashing function is used to disperse the input keys into different buckets. The main advantages are: (1) it can be parallelized

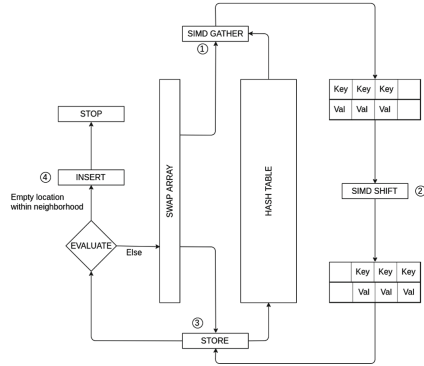


Fig. 5. SIMD accelerated hopscotch hashing - reverse probe

easily, (2) provides a good balance for arithmetic progression values (e.g., primary key). We use Knuth’s multipliers in the hashing function [8].

Aggregation: For simplicity, we perform count as the aggregation function to be performed over the given set of keys. This can be easily extended to perform other aggregation operations such as max, min, sum. The approach fails for other dual pass functions such as average or standard deviation. In this case, normal two pass algorithm is used as in first pass total count is recorded followed by the respective function in the second pass.

All the experiments are run with an increasing number of keys and we record the CPU processing time for complete computation of grouped-aggregation for the different distributions. Every experiment is executed for 20 iterations and the results are averaged. We have performed two different tests over the hashing techniques. In Sect. 5.2, we discuss in detail the impact of the different parameters and we discuss in detail the impact of different distributions on these hashing functions.

The techniques are subjected to insertion with different distribution generators for our test cases. Each distribution represents the characteristics of the input keys provided to the hashing techniques. We synthesize these input based on descriptions given by Gray et al. [5]. The distributions used are (1) unique random, (2) uniform random, (3) moving cluster, (4) exponential, (5) self similar, and (6) heavy hitter. We use the parameters for the generation of the different distributions based on the values given by Gray et al. [5]. For unique random distribution, the seed for generation is selected based on the given input size. We use the random generator function to generate the keys for the uniform random generation. In exponential distribution, the lambda is set to as 0.5, where the number of keys is normalized. Our heavy hitter distribution produces 50% of the given input size as duplicates with remaining keys unique. Finally, the self similar distribution generates 80% of the given input size as duplicates and the remaining 20% is generated as unique keys.

5.2 Factors Affecting Cuckoo Hashing and Hopscotch Hashing

The performance of cuckoo and hopscotch hashing relies on the parameter values, swap number, and neighborhood size respectively. The maximum efficiency of these techniques is achieved by selecting the right value for these parameters. We vary these parameters and find their impact on the load factor. Load factor is the ratio of number of slots filled to the total number of slots available in a hash table. Since, both the hash tables in the worst case cannot accommodate 100% of the hash table slots, we use the parameter for which the load factor is the maximum.

Cuckoo Hashing. We depict in Fig. 6, how the swapping threshold impacts the achievable load factor. Since reaching the swap number threshold requires the table to be rehashed, the execution time of the hashing technique is directly related to the swap number and its corresponding load factor.

For this experiment, we iterate the insertion of keys with size equal to the size of the hash table. We vary the swap number in steps until the total key size. We are able to achieve a maximum load factor of 98.385% for swap number at 25% of the total number of keys. We were not able to achieve hundred percent of the load factor due to the dispersion of keys by the underlying hash function.

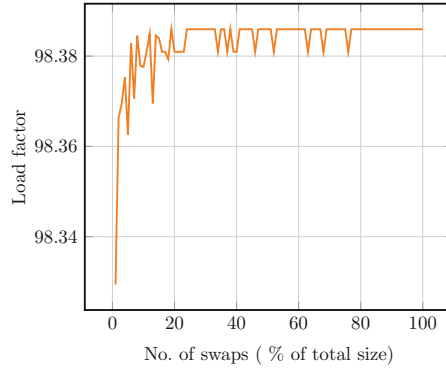


Fig. 6. Cuckoo hashing - swap threshold vs. load factor

Hopscotch Hashing. Hopscotch hashing has minimal insertion time for a key within the neighborhood. Whereas, swapping is performed outside the neighborhood swapping for insertion. Thus, decreasing the neighborhood size leads to faster probing and slower insertion and increasing the neighborhood size does the vice versa. Hence, an optimal neighborhood size must be used for efficient execution. Similar to cuckoo hashing, we evaluate the optimal neighborhood of the hashing technique with different neighborhood sizes and the charts are plotted in Fig. 7.

We run two tests to determine the correct neighborhood size for hopscotch hashing. In our first test, we determine the neighborhood size for optimal insertion time. We vary the neighborhood size from 10% to 100% the total size of the table and record the overall average insertion time for the different sizes. From the results plotted in Fig. 7(a), we observe the lowest insert time is for a neighborhood size 20% of the total table size. We also see that the insert time increases rapidly after the neighborhood size 40% of the total table size mainly due to the multiple swaps for every insertion. Hence, a neighborhood size between 10% to 20% of the total size shows good performance for the hashing technique.

In our second experiment, we investigate the impact of neighborhood size on the load factor. In this experiment, we vary the neighborhood size and record the maximum load factor reached. This is plotted in Fig. 7(b). From our observations, we are able to reach an average of 99% load factor. However, increasing the neighborhood size impacts in the runtime as the number of probe locations increases. We observe that a load factor of 98% is reached for neighborhood size after 20% of the total size for all the data sizes. Hence, the best neighborhood size lies between 20% to 30% of the total size. The remaining keys are not inserted due to poor dispersion of the keys by the hashing function.

Summary. Using the above mentioned parameters, we could reach a maximum load factor of 98% for these two hashing techniques after which re-hashing of the tables is probable. We set the number of swaps for cuckoo hashing as 25 and

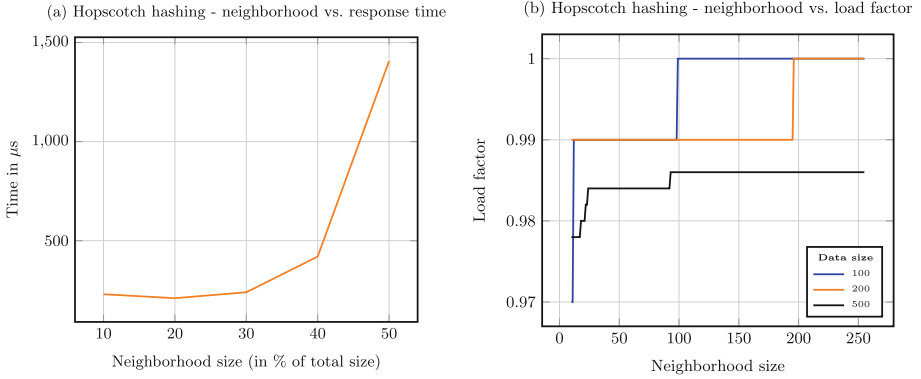


Fig. 7. Hopscotch hashing - impact of neighborhood

the neighborhood size of hopscotch hashing as 50 for all the experiments below. Also, we keep the load factor as 96%.

Impact of Different Distributions. In this experiment, we aggregate keys of various distributions using the hashing techniques with their optimal parameters. Since cuckoo hashing is having a maximum load factor of 98%, we set our maximum load factor as 95% in order to insert all the given keys. Also, we use the optimal parameters for cuckoo and hopscotch hashing.

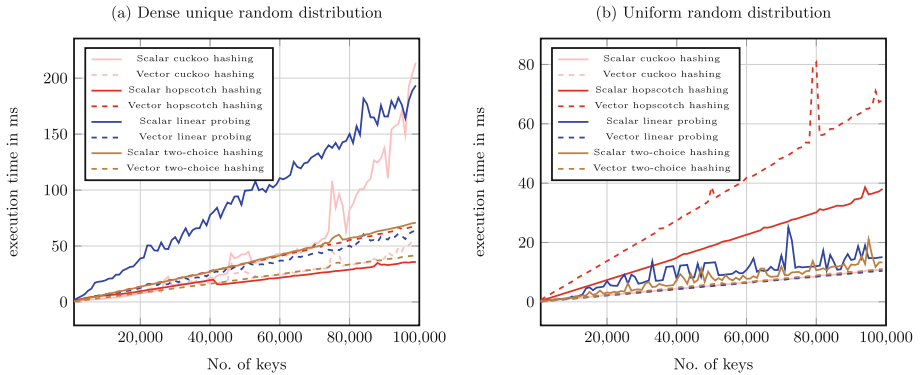


Fig. 8. Hopscotch hashing - impact of distribution

Dense Random Unique Distribution: For unique random distribution linear probing performs worst, whereas the vectorized version competes with other techniques as shown in Fig. 8(a). This is mainly due to the advantage of probing multiple slots in a single step. Cuckoo hashing performance degrades with

increasing data size, mainly due to swaps during insertion. The other hashing techniques have a linear increase of runtime as they have no additional overhead in inserting keys. Specifically, hopscotch hashing works efficiently for this distribution due to the neighborhood size reducing the number of keys to be probed.

Uniform Distribution: Figure 8(b) shows the performance graph for uniform distribution. Hopscotch hashing performs worse in this case due to the overhead of swapping. Cuckoo hashing has nearly similar efficiency as the hopscotch hashing, due to again the penalty from insertion. In case of linear probing and two-choice hashing, the efficiency depends on the order of hashing keys, as the best order of insertion needs less probing. Hence, the runtime oscillates from low to high. However, the vectorized version of the hashing techniques has near linear runtime, except for hopscotch hashing. All other vectorized versions are more efficient than their scalar version, with linear probing having the best speed-up of 3x the scalar version mainly due to an early detection of an empty slot during the probe.

Moving Cluster: We see a peculiar impact on cuckoo hashing for the moving cluster distribution shown in Fig. 9. It performs efficiently until it reaches 50K and after that its performance degrades rapidly. This is mainly due to the heavy swapping of keys inside the cluster. Whereas, the other hashing techniques have a linear performance with increasing data size.

Other Distributions: We omit a performance graph for the other distributions, as the technique’s behavior are similar to what we see for uniform distributed values. The only notable difference is that vectorized cuckoo has good performance for self-similar and exponential distributions for earlier data sizes but is soon surpassed by vectorized two-choice hashing.

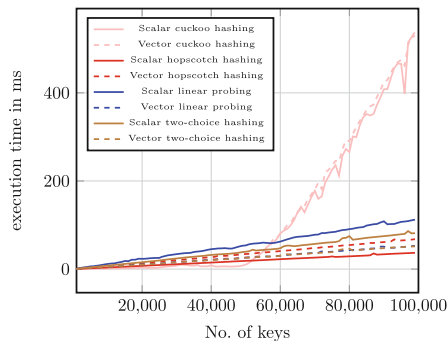


Fig. 9. Moving cluster distribution

Speed-Up Gain. Based on our results, we found that several factors influence the speed-up gained by vectorization of the hashing techniques. We plot the speed-up of the hashing techniques in Fig. 10.

We observe from the figure, linear probing (LP) has consistently positive impact of SIMD acceleration with the maximum of up to 3.7x the speed of scalar version. Two-choice hashing (TCH) also has a considerable impact of vectorization with the maximum gain of 2.5x,

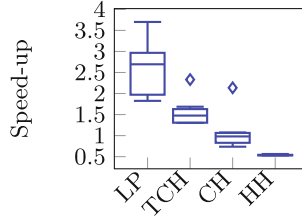


Fig. 10. Speed-up across all tested distributions

but the speed-up depends on the input data distributions. SIMD vectorization has no impact on cuckoo hashing (CH) as the insert time for cuckoo hashing balances the speed-up gained due to probing of keys. However, for distributions with a high number of duplicate keys, the technique gains a speed-up of 2x the scalar version. Finally, our result indicates that the usage of SIMD instructions for hopscotch hashing (HH) is not providing a performance increase. This is due to the preprocessing steps needed for inserting keys. Mainly for SIMD insertion, single key insertion re-arranges multiple keys inside the table.

6 Conclusion

Vectorization impacts the execution of a grouped aggregation function. This impact differs based on the hashing technique used to compute the results. In the work, we explored the impacts on vectorizing the commonly used open-addressing hashing techniques.

For vectorization of the techniques, we provide a framework with horizontal vectorization along with an interleaved insertion. In our method, a given key is searched in a hash table using vectorized probing with either an insertion of a key or an update of the aggregate payload. We detail the execution flow of vectorized hashing techniques and discuss the complexities in incorporating them.

In our experiments, we found that the overhead of vectorizing a scalar key limits the overall performance gain from SIMD. For the case of linear probing and two choice hashing, SIMD acceleration provides gain of 2x with respect to their scalar implementation. Whereas in case of cuckoo hashing and hopscotch hashing, we get negative impact from SIMD. This is mainly due to additional overheads for key insertions in these techniques.

Using our framework, we provide a possible vectorization model for hashing techniques. This model is extensible for further techniques as well as other vectorization strategies. Our current framework is not scalable for a multi-CPU system but a synchronization mechanism can be easily added. Finally, from our analysis, we found that the hashing technique related parameters must be tuned for efficient execution.

Acknowledgments. This work was partially funded by the DFG (grant no.: SA 465/51-1 and SA 465/50-1).

References

1. Broneske, D., Meister, A., Saake, G.: Hardware-sensitive scan operator variants for compiled selection pipelines. In: *Datenbanksysteme für Business, Technologie und Web (BTW)*, pp. 403–412 (2017)
2. Broneske, D., Saake, G.: Exploiting capabilities of modern processors in data intensive applications. *IT - Inf. Technol.* **59**(3), 133 (2017)
3. Cieslewicz, J., Ross, K.A.: Adaptive aggregation on chip multiprocessors. In: *Proceedings of the Very Large Databases (VLDB)*, pp. 339–350 (2007)
4. Flajolet, P., Poblete, P., Viola, A.: On the analysis of linear probing hashing. *Algorithmica* **22**(4), 490–515 (1998)
5. Gray, J., Sundaresan, P., Englert, S., Baclawski, K., Weinberger, P.J.: Quickly generating billion-record synthetic databases. In: *International Conference on Management of Data (SIGMOD)*, pp. 243–252 (1994)
6. Herlihy, M., Shavit, N., Tzafrir, M.: Hopscotch hashing. In: Taubenfeld, G. (ed.) *DISC 2008. LNCS*, vol. 5218, pp. 350–364. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87779-0_24
7. Jiang, P., Agrawal, G.: Efficient SIMD and MIMD parallelization of hash-based aggregation by conflict mitigation. In: *Proceedings of the International Conference on Supercomputing (ICS)*, pp. 24:1–24:11 (2017)
8. McClellan, M.T., Minker, J., Knuth, D.E.: The art of computer programming, vol. 3: sorting and searching. *Math. Comput.* **28**(128), 1175 (1974)
9. Pagh, R., Rodler, F.F.: Cuckoo hashing. *J. Algorithms* **51**(2), 122–144 (2004)
10. Polychroniou, O., Raghavan, A., Ross, K.A.: Rethinking SIMD vectorization for in-memory databases. In: *Proceedings of the International Conference on Management of Data (SIGMOD)*, pp. 1493–1508 (2015)
11. Polychroniou, O., Ross, K.A.: High throughput heavy hitter aggregation for modern SIMD processors. In: *Proceedings of the Ninth International Workshop on Data Management on New Hardware (DaMoN)*, pp. 6:1–6:6 (2013)
12. Richa, A.W., Mitzenmacher, M., Sitaraman, R.: The power of two random choices: a survey of techniques and results. *Comb. Optim.* **9**, 255–304 (2001)
13. Richter, S., Alvarez, V., Dittrich, J.: A seven-dimensional analysis of hashing methods and its implications on query processing. In: *Proceedings of the Very Large Databases (VLDB)*, vol. 9, no. 3, pp. 96–107 (2015)
14. Ross, K.A.: Efficient hash probes on modern processors. In: *Proceedings of the International Conference on Data Engineering (ICDE)*, pp. 1297–1301. IEEE (2007)
15. Zhou, J., Ross, K.A.: Implementing database operations using SIMD instructions. In: *International Conference on Management of Data (SIGMOD)*, p. 145 (2002)



Selecting Sketches for Similarity Search

Vladimir Mic¹, David Novak^{1(✉)}, Lucia Vadicamo², and Pavel Zezula¹

¹ Masaryk University, Brno, Czech Republic

david.novak@fi.muni.cz

² CNR-ISTI, Pisa, Italy

Abstract. Techniques of the Hamming embedding, producing bit string sketches, have been recently successfully applied to speed up similarity search. Sketches are usually compared by the Hamming distance, and applied to filter out non-relevant objects during the query evaluation. As several sketching techniques exist and each can produce sketches with different lengths, it is hard to select a proper configuration for a particular dataset. We assume that the (dis)similarity of objects is expressed by an arbitrary metric function, and we propose a way to efficiently estimate the quality of sketches using just a small sample set of data. Our approach is based on a probabilistic analysis of sketches which describes how separated are objects after projection to the Hamming space.

1 Introduction

Efficient search for data objects according to their pairwise similarity presents an important task in data processing. We consider similarity search for complex objects, e.g. multimedia files. Features of these objects are typically characterized by *descriptors*, which are often high dimensional vectors. They can be bulky and evaluation of their pairwise similarity may be computationally demanding. Thus techniques to process them efficiently are needed. We consider one to one mapping between objects and descriptors, thus we do not distinguish these terms and we use just term object.

Techniques transforming data objects to smaller objects are often used to speed up similarity search. Their number and number of their inherent parameters make their fair comparison difficult. Moreover, the ability of particular approaches to approximate similarity relationships between objects is data dependent. This paper considers a particular family of techniques – transformation of objects to the Hamming space – and it provides formal analysis which allows to efficiently estimate a quality of particular transformation techniques. Our approach uses a small sample set of the original and the transformed objects, and, inspired by the *separability*, which is traditionally used in data clustering and processing of biometrics, we estimate the ability of the transformed objects to distinguish different values of genuine similarity. We define the problem precisely in the following section.

Table 1. Notation used throughout this paper

$(D, d); X \subseteq D$	Metric space with domain D and distance function d ; dataset X
$iDim$	Intrinsic dimensionality of dataset
$sk(o)$	Sketch of object $o \in X$
λ	Length of sketches in bits
$h(sk(o_1), sk(o_2))$	Hamming distance of sketches $sk(o_1)$ and $sk(o_2)$
β	Balance of bits of sketches
$p(x, b)$	Probability that $h(sk(o_1), sk(o_2)) = b$ for $o_1, o_2 \in X : d(o_1, o_2) = x$
$p_i(x, 1)$	Probability $p(x, b)$ for $\lambda = 1$ considering an average bit i
x, b	Values of the distance functions d and h , respectively
Γ	Maximum distance x
ϕ	Number of degrees of freedom of distance function $d(o_1, o_2)$
μ, σ^2	Mean value and variance of Hamming distance
m, s^2	Mean value and variance of probability $p(x, b)$ for a given x

1.1 Problem Formulation

We focus on similarity search in the *metric space* [18]. The notation used throughout the paper is provided in Table 1. Formally, the metric space is a pair (D, d) where D is a domain of objects and d is a total *distance function* $d : D \times D \mapsto \mathbb{R}$. This function determines the dissimilarity of objects – the bigger the value $d(o_1, o_2)$, the less similar the objects $o_1, o_2 \in D$. The distance can be an arbitrary function which satisfies the properties of non-negativity, identity, symmetry and triangle inequality [16].

Having a metric space (D, d) , we consider a dataset $X \subseteq D$, and a *sketching technique* $sk : D \mapsto \{0, 1\}^\lambda$, which transforms objects $o \in D$ to bit-strings of fixed length λ . We call these bit strings *sketches*, and we assume that dissimilarity of these sketches is measured by the *Hamming distance*. Further, we focus on a family of sketching techniques which produce bits *balanced to* β :

- Bit i is balanced to ratio β (with respect to the dataset X) iff it is set to 1 in $\beta \cdot |X|$ sketches $sk(o), o \in X$.

We consider just values $0.5 \leq \beta \leq 1$, since if β is smaller than 0.5 for some bit i , this bit can be flipped in all sketches $sk(o), o \in X$ which preserves all the Hamming distances. The objective of this paper is to propose a way to estimate ability of sketches to approximate similarity relationships between objects $o \in X$, using just a small sample set of data and sketches.

1.2 Related Work

Several sketching techniques have been proposed [1, 5, 7, 10–14, 17] and most of them produce sketches with bits balanced to 0.5. To the best of our knowledge,

there is no prior work which would efficiently estimate, what sketches, their balance β and length λ are suitable for particular data (on condition that β is tunable). For instance, Wang et al. [17] provide analysis to estimate recall of k NN queries for particular sketching technique to select suitable length of sketches. However, their method is not extendible for other sketching techniques and the estimation is rather approximate. Mic et al. also provide approach to estimate suitable length λ for their sketching technique [12], but their ideas cannot be applied for arbitrary sketches. Our proposal is inspired by Daugman’s analysis [3], who investigates binary codes of human irises. He evaluates *separability* of two distance densities to describe the quality of his method to identify people according to this biometric.

The paper is organized as follows. Section 2 contains analysis to estimate an ability of sketches to approximate similarity relationships between objects, Sect. 3 proposes another approach to estimate this quality, Sect. 4 contains discussion about a cost of estimations and results of experiments to compare measured and estimated quality of sketches, and Sect. 5 concludes the paper.

2 Analysis to Estimate Quality of Sketches

The goal of this section is to derive formula describing the ability of sketches to separate two distances from the original metric space. In particular, we consider four arbitrarily selected objects $o_1, o_2, o_3, o_4 \in X$ and their distances $d(o_1, o_2) = x_1, d(o_3, o_4) = x_2, x_1 \leq x_2$. The goal of function $sep_{sk}(x_1, x_2)$ is to describe how separated are the Hamming distances $h(sk(o_1), sk(o_2))$ and $h(sk(o_3), sk(o_4))$.

2.1 A Single-Bit Sketch

We start with an average probability $p_i(x, 1)$ that one bit i of sketches $sk(o_1)$ and $sk(o_2)$ has different value for objects $o_1, o_2 \in X$ with distance $d(o_1, o_2) = x$. This probability can be derived in an analytic way just for some specific sketching techniques [17]. Therefore, we propose to determine it empirically by its evaluation on a sample set of data. We measure the probability $p_i(x, 1)$ on equidistant intervals of distances x . To make function $p_i(x, 1)$ continuous, we use linear interpolation between measured points and we add an artificial point $[0, 0]$ to catch influence of smaller distances than were observed on the sample set. We work with an average probability $p_i(x, 1)$ evaluated over all bits i . Probability function $p_i(x, 1)$ constitutes one of features describing quality of sketches, as it should obviously increase with x . An example is provided in Fig. 1a.

2.2 Projection of Distance x on Hamming Distance b

As a next step, we derive probability function $p(x, b)$ that Hamming distance $h(sk(o_1), sk(o_2))$ is equal to b for objects o_1, o_2 with distance $d(o_1, o_2) = x$. It is done by composition of λ instances of probability function $p_i(x, 1)$. This step is challenging due to possible bit correlations. Probability function $p(x, b)$ for a

fixed x can be modelled by a symmetric function¹, which allows us to use its *binomial analogue*. It is a scaled binomial distribution with the same variance as function $p(x, b)$. To fit variance of function $p(x, b)$, we need to estimate its *number of degrees of freedom* ϕ [3].

Lemma 1. *The number of degrees of freedom ϕ of function $p(x, b)$ is similar to the number of degrees of freedom ϕ' of the density of the Hamming distance on all sketches $sk(o), o \in X$.*

Clarification. Daugman [3] evaluates the number of degrees of freedom of the density of the Hamming distance on sketches:

$$\phi' = \frac{\mu \cdot (\lambda - \mu)}{\sigma^2} \quad (1)$$

where λ is the length of sketches, μ is the mean value and σ^2 is the variance of the Hamming distance. According to analysis in [13], the μ is given by λ and β , and σ^2 is given by λ , β and pairwise bit correlations. Therefore, iff sketches of objects $o_1, o_2 : d(o_1, o_2) = x$ have bits balanced to β and they have same pairwise bit correlations as all the sketches $sk(o), o \in X$, the Lemma 1 describes equality, i.e. $\phi = \phi'$. Our first approach to estimate quality of sketches assumes this equation, and the error caused by this assumption is discussed in Sect. 2.4.

We connect Eq. 1 with the term *intrinsic dimensionality* ($iDim$), which describes an amount of information in data. Several ways to estimate the $iDim$ have been developed but just a few of them can be used in a general metric space. We use the formula of Chávez and Navarro [2]:

$$iDim \approx \frac{\mu^2}{2 \cdot \sigma^2}. \quad (2)$$

The mean value μ equals $2\lambda \cdot \beta \cdot (1 - \beta)$ [13], and thus, using the Eqs. 1, 2 and Lemma 1, we may express the number of degrees of freedom ϕ using the intrinsic dimensionality of sketches $iDim$ and balance of their bits β :

$$\phi = \frac{\mu \cdot (\lambda - \mu)}{\sigma^2} = \frac{\mu \cdot \frac{\mu}{2 \cdot \beta \cdot (1 - \beta)}}{\sigma^2} - \frac{\mu^2}{\sigma^2} \approx 2 \cdot iDim \cdot \left(\frac{1}{2 \cdot \beta \cdot (1 - \beta)} - 1 \right). \quad (3)$$

In order to model probability $p(x, b)$, we propose to use binomial distribution with ϕ degrees of freedom which we scale and interpolate to get the final function. The only input necessary for the usage of this binomial analogue is $iDim$ of sketches, empirically evaluated on a sample set of sketches, and balance of their bits β . We round number of degrees of freedom to the nearest integer and denote it ϕ in the rest of this paper. In the following, we describe the estimation of $p(x, b)$ formally. We approximate this function by a linear interpolation $p_{lin}(x, b)$ normalized with a coefficient $coef(x)$:

$$p(x, b) \approx \frac{p_{lin}(x, b)}{coef(x)}. \quad (4)$$

¹ Reasoning is provided at <https://www.fi.muni.cz/~xmic/sketches/Symmetry.pdf>.

The normalization coefficient $coef(x)$ is evaluated as:

$$coef(x) = \sum_{i=0}^{\phi} p_{lin}(x, i)$$

and the linear interpolation $p_{lin}(x, b)$ as:

$$p_{lin}(x, b) = p_{int}(x, \lfloor b' \rfloor) + (b' - \lfloor b' \rfloor) \cdot (p_{int}(x, \lceil b' \rceil) - p_{int}(x, \lfloor b' \rfloor))$$

where b' is the scaled value b :

$$b' = b \cdot \frac{\phi}{\lambda}$$

and where function $p_{int}(x, b_{int})$ (which requires the second parameter to be an integer), evaluates the binomial distribution:

$$p_{int}(x, b_{int}) = \binom{\phi}{b_{int}} \cdot p_i(x, 1)^{b_{int}} \cdot (1 - p_i(x, 1))^{\phi - b_{int}}.$$

We use linear interpolation p_{lin} since b' is usually not an integer. Due to the transformation of the binomial distribution, it is necessary to normalize probability using $coef(x)$ coefficient: we normalize function p_{int} by the sum over all values, since $p(x, b)$ is discrete with respect to b .

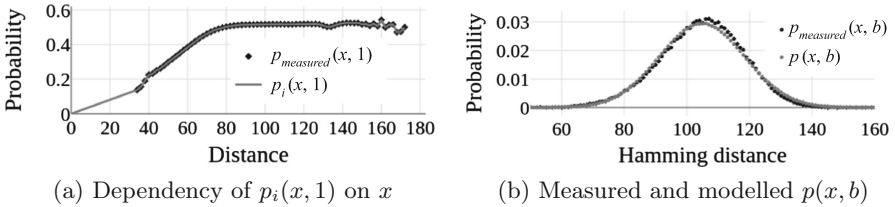


Fig. 1. Example of functions $p_i(x, 1)$ and $p(x, b)$ used in the model

We show an example, how this binomial analogue fits the distribution of the measured $p(x, b)$ in Fig. 1b. The black points show values empirically measured, and grey points show values estimated by the proposed binomial analogue described by Eq. 4. In this experiment, we used sketches with $\lambda = 205$ bits balanced to $\beta = 0.5$, and just sketches of objects within distance $x = 86$ evoking probability $p_i(x, 1) = 0.51$. These measurements confirm that the binomial analogue is a good approximation of probability $p(x, b)$.

2.3 Quality of Sketches

Quality of sketches used for the similarity search is given by their ability to preserve similarity relationships between objects. Let us consider four arbitrarily selected objects $o_1, o_2, o_3, o_4 \in X$ and distances $d(o_1, o_2) = x_1$ and

$d(o_3, o_4) = x_2, x_1 \leq x_2$. In the following, we focus on a *separation* of probability functions $p(x_1, b)$ and $p(x_2, b)$ and we describe it by formula adopted from work of Daugman [3]:

$$sep_{sk}(x_1, x_2) = \frac{m_2 - m_1}{\sqrt{\frac{s_1^2 + s_2^2}{2}}} \tag{5}$$

where m_1 and m_2 are mean values of $p(x_1, b)$ and $p(x_2, b)$, and s_1^2 and s_2^2 are their variances. These values can be expressed by analysis of Eq. 4:

Lemma 2. *Mean value m of probability function $p(x, b)$ is $p_i(x, 1) \cdot \lambda$. Variance s^2 of probability function $p(x, b)$ is $\frac{\lambda^2}{\phi} \cdot p_i(x, 1) \cdot (1 - p_i(x, 1))$.*

Proof. Function $p(x, b)$ is formed by binomial distribution (see function p_{int} in Eq. 4), which is scaled with respect to value b by coefficient $\frac{\phi}{\lambda}$. Since mean of function p_{int} is $p_i(x, 1) \cdot \phi$ and its variance is $\phi \cdot p_i(x, 1) \cdot (1 - p_i(x, 1))$, then:

$$m = p_i(x, 1) \cdot \phi \cdot \frac{\lambda}{\phi} = p_i(x, 1) \cdot \lambda,$$

and

$$s^2 = \phi \cdot p_i(x, 1) \cdot (1 - p_i(x, 1)) \cdot \left(\frac{\lambda}{\phi}\right)^2 = \frac{\lambda^2}{\phi} \cdot p_i(x, 1) \cdot (1 - p_i(x, 1)).$$

Theorem 1. *Considering four arbitrary objects $o_z \in X, z \in [1..4]$ with distances $d(o_1, o_2) = x_1, d(o_3, o_4) = x_2, x_1 \leq x_2$, and an arbitrary sketching technique sk producing sketches with bits balanced to β , the separation $sep_{sk}(x_1, x_2)$ of the Hamming distances $h(sk(o_1), sk(o_2))$ and $h(sk(o_3), sk(o_4))$ can be expressed:*

$$sep_{sk}(x_1, x_2) \approx 2 \cdot \sqrt{iDim} \cdot f_{sk}(x_1, x_2) \cdot \sqrt{\frac{1}{2 \cdot \beta \cdot (1 - \beta)} - 1} \tag{6}$$

where

$$f_{sk}(x_1, x_2) = \frac{p_i(x_2, 1) - p_i(x_1, 1)}{\sqrt{p_i(x_1, 1) \cdot (1 - p_i(x_1, 1)) + p_i(x_2, 1) \cdot (1 - p_i(x_2, 1))}} \tag{7}$$

Proof. Theorem holds as a consequence of Eqs. 3, 5, and Lemma 2.

Theorem 1 reveals features of sketches $sk(o), o \in X$, which improves their capability to approximate similarity relationships between objects. For instance, sufficiently high $iDim$ of sketches is necessary to allow them distinguish distances $d(o_1, o_2) < d(o_3, o_4)$. Please notice, that just function $f_{sk}(x_1, x_2)$ (defined by Eq. 7) takes into account values x_1 and x_2 , and that there is no direct dependency of $sep_{sk}(x_1, x_2)$ on sketch length λ .

To describe quality of sketches, we propose to evaluate $sep_{sk}(x_1, x_2)$ over whole range of distances function d . Without loss of generality, we assume that d is continuous and its range is $[0, \Gamma]$. Then:

$$quality(sk) = \int_0^\Gamma \int_{x_1}^\Gamma sep_{sk}(x_1, x_2) \partial x_2 \partial x_1 \tag{8}$$

Semantics of this integral is in compliance with the sign of $sep_{sk}(x_1, x_2)$: value of $sep_{sk}(x_1, x_2)$ is negative iff the distances x_1, x_2 are swapped after the transformation to sketches². Such distances x_1, x_2 then naturally decrease the quality of sketching technique, as described by this equation. Since $quality(sk)$ cannot be meaningfully compared for metrics with different Γ , we propose to normalize it. Equation 8 allows direct normalization by Γ^2 . Finally, if sketches are going to be applied for similarity search, they are needed to well separate small distances from the others. Therefore, it is meaningful to evaluate quality of sketching technique using some threshold $t < \Gamma$:

$$quality_{norm}(sk, t) = \frac{\int_0^t \int_{x_1}^{\Gamma} sep_{sk}(x_1, x_2) \partial x_2 \partial x_1}{\Gamma^2}. \quad (9)$$

Therefore, we propose to evaluate $quality_{norm}(sk, t)$ using a sample set of data and use it to compare quality of sketches. The cost of this estimation is discussed in Sect. 4.1.

2.4 Sources of Error

The main source of error of proposed quality estimation is caused by Lemma 1, as we assume that balance β and pairwise correlations of bits of sketches $sk(o_1), sk(o_2), o_1, o_2 \in X : d(o_1, o_2) = x$ for an arbitrary given x are the same, as on the whole dataset X . The precision of this assumption is data and sketching technique dependent. We have observed, that the proposed binomial analogue is quite precise for non-extreme distances x (as shown by Fig. 1b), but its precision decreases mainly for tails of x . Therefore, this feature causes an erroneous estimation for some sketching techniques and datasets. An example is given in Fig. 2, where the binomial analogue $p(x, b)$ for a very low x is examined.

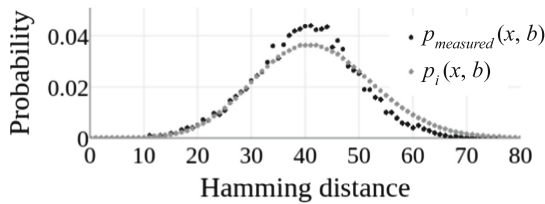


Fig. 2. Measured and modelled $p(x, b)$ for $x = 38$ implying $p_i(x, 1) = 0.2$

Another errors are caused by our intention to use a small sample set to evaluate $quality_{norm}(sk, t)$, and by the fact that the provided analysis is based on expected values and it does not consider deviations from modelled functions.

² Please see, that the sign of $sep_{sk}(x_1, x_2)$ is given by the sign of function $f_{sk}(x_1, x_2)$, and this is negative iff $p_i(x_2, 1) < p_i(x_1, 1)$. We have assumed $x_1 \leq x_2$, and these two inequalities are equivalent to swapping distances x_1, x_2 .

We evaluate experiments with this approach in Sect. 4, and we denote it *Approach A* (as *analytique*). In the following section, we propose the second way to estimate quality of sketches, which aims to mitigate the error of Approach A.

3 Approach PM

The second way to estimate $quality_{norm}(sk, t)$ is based on direct evaluation of means m_1, m_2 and variances s_1^2, s_2^2 of the $p(x, b)$, used in Eq. 5. If these values are evaluated directly on a sample set of data, just this equation and Eq. 9 are utilized to estimate $quality_{norm}(sk, t)$. However, this approach requires evaluation of mean m and variance s^2 of $p(x, b)$ for the whole range of distances $x \in [0..T]$.

In particular, we use equidistant intervals of x and we evaluate distances $b = h(sk(o_1), sk(o_2))$ for each pair of objects o_1, o_2 from the sample set such that $d(o_1, o_2)$ is from a given interval of x . Then we evaluate the mean m and variance s^2 for each interval of x and we add an artificial mean $m = 0$ and variance $s^2 = 0$ for distance $x = 0$. Finally, we use linear interpolation to get values m and s^2 for an arbitrary distance x .

We denote the estimation of $quality_{norm}(sk, t)$ according to this procedure *Approach PM* (*partially measured*) and we evaluate its capability to estimate quality of sketches in Sect. 4. At first, let us discuss sources of errors of this approach. The cost of this estimation is discussed in Sect. 4.1.

3.1 Sources of Error of the PM Approach

Approach PM mitigates an error brought to the Approach A by too strong usage of Lemma 1. Instead of this error, Approach PM is more sensitive on a low number of objects o_1, o_2 within very small distances $d(o_1, o_2) = x$ in the sample set. Since the mean m and variance s^2 of $p(x, b)$ is examined for the whole range of distance x , it needs a representative number of objects o_1, o_2 within each interval of x to measure it precisely, which is obviously a problem for tails of distances x . However, exactly the ability to precisely handle extremely small distances is crucial to well estimate quality of sketches for similarity search.

The rest of errors is caused by similar features as in case of Approach A (see Sect. 2.4 for details). Therefore, both approaches provide estimation with some probable level of error, and we evaluate them both in the next section.

4 Experiments

This section provides verification of the proposed approaches to estimate quality of sketches. At first, we discuss costs of proposed estimations in comparison with a traditional approach to evaluate quality of sketches.

4.1 Queries Evaluations vs. Proposed Estimations

Testing quality of sketches is usually performed via evaluation of sufficient number of representative queries on sample data. Therefore, the precise answer for these queries must be known, and the sketches for both, the dataset and query set must be created.

We use the recall of k nearest neighbours queries (k NN) evaluated via simple sketch-based filtering. In particular, sketches are applied to *filter* a fixed number of the most similar sketches to the query sketch $sk(q)$, and then the set of corresponding objects $CandSet(q)$ is *refined* by the distances $d(q, o), o \in CandSet(q)$. Finally, the k most similar objects from the $CandSet(q)$ are returned as the query answer. The *recall* expresses the relative intersection of this answer with the precise answer returned by the sequential evaluation of all distances $d(q, o), o \in X$. Please, notice that a suitable size of the $CandSet(q)$ considering applied sketching technique must be selected, which is another difficult and data dependent task. Finally, even if this expensive procedure is performed, it is relevant just for a tested dataset and it is dependent on a selection of queries.

We evaluate one thousand 100NN queries on two datasets of size 1 million objects. Therefore, both ground truths cost 2 billion evaluations of distance $d(q, o)$, several (30) different sets of million sketches are created, and finally, billion Hamming distances are evaluated for each set of sketches. In our experiments, we use 16 and 14 different sets of sketches for our datasets. The cost of their creation is in order of billions of distance computations, and several GB of data are read from hard-drives during these experiments.

Conversely, proposed quality estimations do not use any queries. We use just a sample set of 5000 objects and their sketches in our experiments. In case of Approach A, we use 2 million distances to get function $p_i(x, 1)$ and $iDim$ of sketches. The efficiency of the estimation is given mainly by the precision of integral evaluation (defined by Eq. 9). Since we use parameters providing high precision, an evaluation of $quality_{norm}(sk, t)$ by Approach A for one set of sketches takes about 50s on average. Approach PM is even more efficient, as it uses 2 millions distances to get means m and variances s^2 of $p(x, b)$ directly. Its evaluation takes approximately 30s per set of sketches on average.

4.2 Test Data

We use two real-life datasets, both consisting of visual descriptors extracted from images. The first one is formed by 1 million *DeCAF* [4, 15] descriptors from the *Profiset collection*³. These descriptors are 4,096-dimensional vectors of float numbers taken as an output from the last hidden layer of a deep convolutional neural net [8]. Although this neural net has been trained for ImageNet classification, the last hidden layer is suitable for solving recognition problems [4, 15]. These descriptors with Euclidean distance L_2 form the metric space (D, d) . The distance density is depicted in Fig. 3a and the intrinsic dimensionality of dataset (defined in Sect. 2.2) is 26.9.

³ <http://disa.fi.muni.cz/profiset/>.

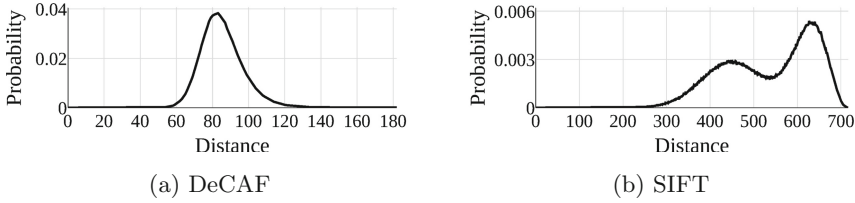


Fig. 3. Distance densities of $d(o_1, o_2), o_1, o_2 \in X$

The second dataset is formed by 1 million *SIFT* descriptors [9] from ANN dataset⁴. These descriptors are 128-dimensional vectors of unsigned integers. We compare them with Euclidean distance as well, and density of this distance is depicted in Fig. 3b. The intrinsic dimensionality of this dataset is 13.4.

4.3 Sketching Techniques

We examine four sketching techniques in this paper. The *GHP_50* technique is adopted from paper [12]. It is based on *generalized hyperplane partitioning* (GHP) depicted in Fig. 4. A pair of pivots $p_{i1}, p_{i2} \in D$ is selected for each bit i of sketches $sk(o), o \in X$, and value of bit i expresses which of these two pivots is closer to o . Therefore, one instance of GHP determines one bit of all sketches $sk(o), o \in X$. The pivot pairs are selected to produce balanced and low correlated bits.

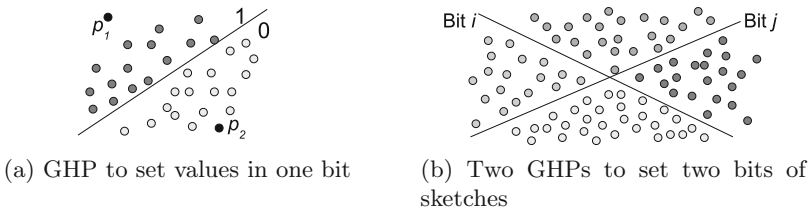


Fig. 4. Generalized hyperplane partitioning for sketching technique

In particular, the pivot selection [12] works as follows: (1) an initial set of pivots P_{sup} is selected at random from domain D , (2) balance of GHP is evaluated using a sample set of X for all pivot pairs $(p_1, p_2), p_1, p_2 \in P_{sup}$, (3) set P_{bal} is formed by all pivot pairs that divide the sample set into two parts balanced with tolerance 5% (at least 45% to 55%) and corresponding sketches sk_{bal} with balanced bits are created. (4) The absolute value of Pearson correlation coefficient is evaluated for all pairs of bits of sketches sk_{bal} to form correlation matrix M , and (5) a heuristic is applied to select rows and columns of M , which form its sub-matrix with low values and size $\lambda \times \lambda$. (6) Finally, the pivot pairs

⁴ <http://corpus-texmex.irisa.fr/>.

which produce the corresponding low correlated bits define sketches $sk(o), o \in X$. A pseudo-code of this heuristic is available online⁵.

The second technique *GHP_80* is similar to *GHP_50*, but the pivots P_{bal} are selected to produce bits balanced to $\beta = 0.8$. This sketching technique have been discussed to produce sketches of the similar quality as *GHP_50*, considering sufficiently high sketch length λ [13]. However, such sketches should be indexable more easily due to their lower intrinsic dimensionality.

Technique *BP_50* uses *ball partitioning* instead of *GHP*. *BP* is defined by a pivot and radius to split data into two parts. We evaluate distances to pivot for each object from a sample set to select radius dividing sample set into halves. Therefore λ pivots are selected to create sketches of length λ . To ensure as small pairwise bit correlations as possible, we employ the same heuristic as in case of techniques *GHP_50* and *GHP_80*.

The last technique *THRR_50* is inspired by papers [6, 10], and it is the only one of examined sketching techniques which is applicable just in vector space, not in the more generic metric space. It uses the principal component analysis (PCA) to shorten vectors to length λ . Then these shortened vectors are rotated using a random matrix, and finally they are binarized using the median values of each their dimension: the bit value expresses, whether the value in a given position of rotated shortened vector is higher or lower then the median evaluated on a sample set. Therefore, the balance β of bits is 0.5. The random rotation of shortened vectors helps to distribute information equally over the vector, as the PCA returns vectors with decreasing importance of values in particular positions. Since the binarization dismiss this different importance, it is suitable to rotate shortened vectors randomly and then binarize [6].

We create sketches $sk(o), o \in X$ of four different lengths: 64, 128, 192 and 256 bits, by each of the sketching technique for the both, DeCAF and SIFT datasets. The only exception is constituted by *THRR_50* on SIFT dataset, as this technique cannot produce sketches longer then the original vectors. Therefore, for this combination we examine just lengths 64 and 128 bits.

4.4 Results

To verify capability of Approaches A and PM to estimate quality of sketches, we compare these estimations with the recall of k NN queries, using the procedure and parameters described in Sect. 4.1. We use *CandSet*(q) of size 2000 objects (i.e. 0.2% of the dataset X), and we depict results by *box plots* to show distribution of values for particular query objects (see Fig. 5 for its definition).

The results for the DeCAF dataset are depicted in Fig. 6. The box-plots, which describe the measured recall use the primary y axis. The names of particular sketching techniques (on x axis) are of the form $skTech_{\beta-\lambda}$ where β is the balance of bits in percentages and λ is the sketch length in bits. Quality estimations use the secondary axis y : Approach A is expressed by dashed line and Approach PM by full line. As we have two estimations of the same feature,

⁵ <https://www.fi.muni.cz/~xmic/sketches/AlgSelectLowCorBits.pdf>.

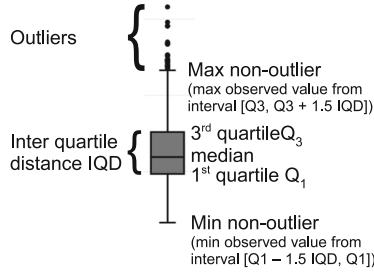


Fig. 5. Box-plot

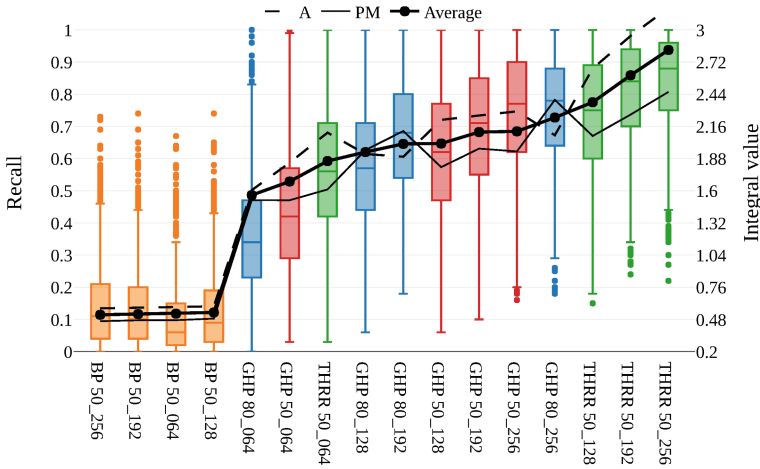


Fig. 6. DeCAF dataset: the measured recall and quality estimations by approaches A, PM and their average

we evaluate even the average of these two estimations, which is expressed by black curve with points.

The results for the SIFT dataset are depicted in the same way in Fig. 7. There is a clear correspondence for both datasets between the recall and estimations. However, in case of SIFT, the Approach PM significantly overestimates sketches THRR and Approach A underestimates it. Another remark is, that the quality of sketching techniques is data dependent, as for instance the BP techniques perform bad on DeCAF dataset but for SIFT they are still reasonable. The interpretation of these results should take into account, that the recall is strongly dependent on the size of the $CandSet(q)$ as well.

We show the Pearson correlation coefficient between the estimated values and the medians of measured recalls in Table 2. The both approaches provide top quality results in case of DeCAF dataset, but the estimations are not as good in case of SIFT due to THRR technique. Nevertheless, the Approach PM

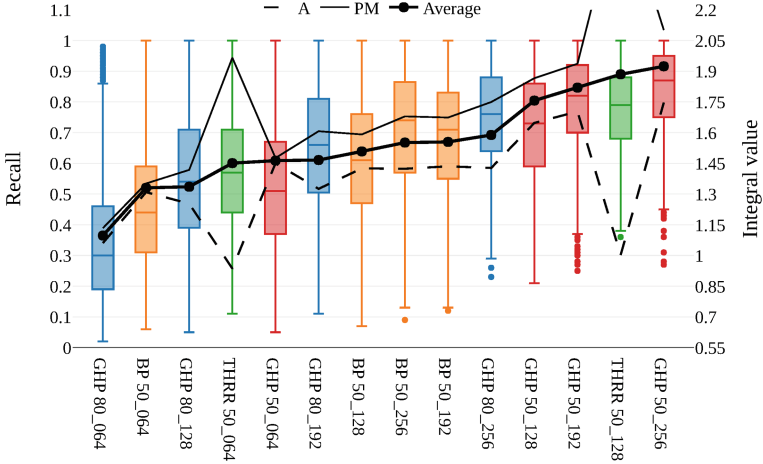


Fig. 7. SIFT: the recall, quality estimations by approach A, PM and their average

Table 2. Correlations of quality estimations and measured medians of the recall

	Approach A	Approach PM	Average
DeCAF	+0.96	+0.97	+0.98
SIFT	+0.55	+0.74	+0.93

still provides a strongly correlated estimation with the measured values, and the quality of averaged estimation is of a top quality even in case of this dataset.

We do not discuss an efficiency of query processing, as we consider sequential evaluation of all Hamming distances $h(sk(q), sk(o))$, $o \in X$ during query evaluation. In this case, query processing times are equal for all sets of sketches, as they are given by the size of candidate set. Indexing of sketches pays off mainly for huge datasets, and its efficiency is influenced by $iDim$ of sketches. However, our preliminary experiments on just two very different datasets do not justify any reasonable conclusions about this feature, and thus we postpone it to the future work.

5 Conclusions

Several techniques of the Hamming embedding have been proposed to speed up similarity search. Since their parameters (including the length of the transformed objects – sketches) must be selected in advance, and their ability to approximate similarity relationships between objects is data dependent, the selection of particular sketches for similarity search is a challenging problem.

We proposed two efficient approaches to estimate the quality of sketches with respect to particular data. These approaches do not need any ground truth or

query evaluations but just a small sample set of data objects and their sketches. Both approaches are based on analytic study of sketches. Experiments with two real-life datasets show that they provide a reasonable estimation of the quality of sketches when compared with the recall of k NN queries. The average of the proposed estimations follow the medians of the measured recall with correlations $+0.98$ and $+0.93$, in cases of our two datasets.

Acknowledgements. Paper was supported by the Czech Science Foundation project GBP103/12/G084.

References

1. Charikar, M.: Similarity estimation techniques from rounding algorithms. In: Proceedings on 34th Annual ACM Symposium on Theory of Computing, Montréal, Québec, Canada, 19–21 May 2002, pp. 380–388 (2002)
2. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. *ACM Comput. Surv.* **33**(3), 273–321 (2001)
3. Daugman, J.: The importance of being random: statistical principles of iris recognition. *Pattern Recogn.* **36**(2), 279–291 (2003)
4. Donahue, J., et al.: DeCaf: a deep convolutional activation feature for generic visual recognition. In: *ICML*, vol. 32, pp. 647–655 (2014)
5. Dong, W., Charikar, M., Li, K.: Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces. In: Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. *ACM* (2008)
6. Gordo, A., Perronnin, F., Gong, Y., Lazebnik, S.: Asymmetric distances for binary embeddings. *IEEE Trans. Pattern Anal. Mach. Intell.* **36**(1), 33–47 (2014)
7. Jegou, H., Douze, M., Schmid, C.: Hamming embedding and weak geometric consistency for large scale image search. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) *ECCV 2008 Part I. LNCS*, vol. 5302, pp. 304–317. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88682-2_24
8. Krizhevsky, A., Sutskever, I., Hinton, G.E.: *Imagenet classification with deep convolutional neural networks*, vol. 60, pp. 84–90 (2017)
9. Lowe, D.G.: Object recognition from local scale-invariant features. In: *ICCV*, pp. 1150–1157 (1999)
10. Lv, Q., Charikar, M., Li, K.: Image similarity search with compact data structures. In: Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, 8–13 November 2004, pp. 208–217 (2004)
11. Lv, Q., Josephson, W., Wang, Z., Charikar, M., Li, K.: Ferret: a toolkit for content-based similarity search of feature-rich data. In: *ACM SIGOPS Operating Systems Review* (2006)
12. Mic, V., Novak, D., Zezula, P.: Designing sketches for similarity filtering. In: *IEEE International Conference on Data Mining Workshops, ICDMW 2016, Barcelona, Spain, 12–15 December 2016*, pp. 655–662 (2016)
13. Mic, V., Novak, D., Zezula, P.: Sketches with unbalanced bits for similarity search. In: Beecks, C., Borutta, F., Kröger, P., Seidl, T. (eds.) *Similarity Search and Applications*, vol. 10609, pp. 53–63. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68474-1_4

14. Muller-Molina, A.J., Shinohara, T.: Efficient similarity search by reducing i/o with compressed sketches. In: Proceedings of the 2nd International Workshop on Similarity Search and Applications, pp. 30–38 (2009)
15. Oquab, M., Bottou, L., Laptev, I., Sivic, J.: Learning and transferring mid-level image representations using convolutional neural networks. In: Proceedings of the IEEE CVPR Conference (2014)
16. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann, Burlington (2006)
17. Wang, Z., Dong, W., Josephson, W., Lv, Q., Charikar, M., Li, K.: Sizing sketches: a rank-based analysis for similarity search. In: Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2007, San Diego, California, USA, 12–16 June 2007, pp. 157–168 (2007). <https://doi.org/10.1145/1254882.1254900>
18. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach, vol. 32. Springer, Heidelberg (2006). <https://doi.org/10.1007/0-387-29151-2>



On the Support of the Similarity-Aware Division Operator in a Commercial RDBMS

Guilherme Q. Vasconcelos^{1(✉)}, Daniel S. Kaster², and Robson L. F. Cordeiro¹

¹ Institute of Mathematics and Computer Sciences,
University of São Paulo, São Carlos, Brazil
gui.queirozv@usp.br, robson@icmc.usp.br

² Computer Science Department, University of Londrina, Londrina, Brazil
dskaster@uel.br

Abstract. The division operator from the relational algebra allows simple and intuitive representation of queries with the concept of “for all”, and thus it is required in many real applications. However, the relational division is unable to support the needs of modern applications that manipulate complex data, such as images, audio, long texts, genetic sequences, etc. These data are better compared by similarity, whereas relational algebra always compares data by equality or inequality. Recent works focus on extending relational operators to support similarity comparisons and their inclusion in relational database management systems. This work incorporates and studies the behavior of several similarity-aware division algorithms in a commercial RDBMS. We compared the two state-of-art algorithms against several SQL statements and found when to use each one of them in order to improve query time execution. We then propose an extension of the SQL syntax and the query analyzer to support this new operator.

Keywords: Database · Relational division · Similarity comparison

1 Introduction

The relational division [1] allows writing queries involving the concept of “for all” in a simple and intuitive manner, being employed in queries performed by real applications. For example: (1) “*What products have all the requirements for the industrial quality control?*”; (2) “*What cities have all the requirements to produce a given type of crop?*”; (3) “*What companies have all the products required in a request for tender?*”. This operator identifies groups of tuples in one relation that share the same values for a given set of attributes, and whose values for the remainder attributes cover all values contained in the tuples of a second relation.

Let us use the first example to describe the division. Consider the existence of an industrial production line that wants to implant an automatic real-time

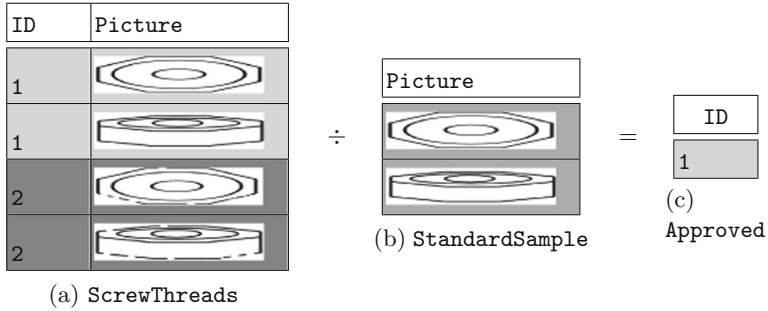


Fig. 1. A toy example illustrating the similarity division to select newly produced screw threads similar to the sampled. Images from group with ID = 2 represent defective and cracked screw threads.

quality control. Thus, a flawless sample is selected to be considered the “standard sample” from which many images are extracted from many different angles and stored in a set. Then, from every newly produced item, images from the same angles are taken and stored in another set. Finally, the quality inspection of a produced item is to check, for every image, if the item’s image is sufficiently similar to the corresponding standard sample’s image. The division operation of the second set (items) by the first set (sample) would return exactly the items that passed the quality control.

However, the relational division can only answer queries by equality whereas images are better compared by similarity. Many researchers have been extending relational operators, such as the division operator, to support similarity comparisons. The similarity-aware division operator is able to answer queries with the concept of “candidate elements and requirements” on distinct but similar data by considering as valid elements with similar attribute value, according to a distance function and a threshold defined by the user. In the example, suppose that the images from the standard sample are stored in a relation named **StandardSample(Picture)**, where **Picture** is an image attribute, as illustrated in Fig. 1b, and the items’ images are stored in relation **ScrewThreads(ID, Picture)**, where **ID** is the identifier of the item from which the picture was taken (Fig. 1a). The result of the division between the relations **ScrewThreads** and **StandardSample** is the relation **Approved** (Fig. 1c), containing the ID of the items that have all their images similar to the corresponding ones from the standard sample, which in the example is only item 1.

Early works regarding the similarity-aware division showed the usability of this operator in real datasets in applications such as: finding relevant request for tenders a company is able to satisfy by comparing their product catalog with the products required [2]; identifying cities able to support a type of crop by comparing satellite images of cities against the requirements’ images; and identifying animals with all genetic sequences similar to animals that have predefined genetic traits, to develop high quality derivative, such as milk, for selective breeding [3].

This article focuses on the inclusion of the similarity-aware division operator in an Relational Database Management System (RDBMS). We propose syntactic representations, implemented different execution algorithms in the Oracle RDBMS, and evaluate their performance on synthetic data that represent possible input relations for this operator. Our main contributions are: (1) SQL syntaxes to represent the Similarity-Aware Division operator; (2) The inclusion of this operator in an commercial RDBMS; (3) Hints from experimental evaluation to help decide when to execute the appropriate algorithm.

The remainder of this article is organized as follows. Section 2 presents the background on similarity queries. Section 3 focuses on the related work regarding relational division and the inclusion of similarity algorithms in RDBMS. Section 4 presents our proposed inclusion of the similarity-aware division operator in a RDBMS. Finally, Sect. 5 discusses the experimental evaluation, and Sect. 6 concludes this article.

2 Background

2.1 Similarity Queries in Metric Spaces

Similarity queries have been widely employed as a mean to manipulate complex data such as audio, video, images, genetic sequences, long texts etc. Such data does not present total order relationship so operators like $<$, \leq , $>$, \geq cannot be used and the identity comparison ($=$) is usually meaningless as these types of data will hardly ever be equal to one another [4]. In this context, complex data are represented in a metric space by their intrinsic feature, i.e., an image can be compared against each other using numeric features that represent color, texture or shape patterns extracted from their visual content. This representation allows evaluating how much an element is similar to another according to a distance function.

Formally, a metric space [4] is a pair defined as $M = \{\mathbb{S}, d\}$ where \mathbb{S} is the data domain and d is the distance function that expresses the similarity between elements of \mathbb{S} by a value between 0 and 1 where values closer to zero means more similar and closer to one less similar. The distance function must satisfy the following properties: symmetry: $d(s_1, s_2) = d(s_2, s_1)$; non-negativity: $0 < d(s_1, s_2) < \infty$ if $s_1 \neq s_2$ and $d(s_1, s_2) = 0$ if $s_1 = s_2$; triangular inequality: $d(s_1, s_2) \leq d(s_1, s_3) + d(s_3, s_2)$ for any $s_1, s_2, s_3 \in \mathbb{S}$. Data in a metric space can be indexed in structures called a Metric Access Method (MAM), that divides the dataset into regions and choses pivots elements to prune the space through triangular inequality accordingly to their distance to the query element [4].

Many researchers have been extending relational operators, such as the **Selection** operator, to support similarity queries based on two approaches: (a) the range query, which selects all elements that are similar enough to a query center element, up to a predefined threshold; and (b) k -Nearest Neighbors (k -NN) query, which returns the k most similar elements to the query center [5].

As well as the *range* and *k-NN* queries extensions to the **Selection** operator, other relational operators have been extended to support similarity, such as the

Similarity Join. There are a number of variants for this operator, usually following the same principles of range and k -NN queries.

There are proposals for the set-based operators as well. Marri et al. [6], provided efficient similarity-aware algorithms for the **Union**, **Intersect** and **Difference** operators for unidimensional data and implemented them on Postgres. Pola et al. [7] presented the concept of **SimSets**, which are sets with no pair of elements that are similar to each other up to a threshold. There exists also algorithms for **Grouping** and **Aggregation** [8].

Finally, the **Similarity-aware Division** operator [3] was recently proposed by Gonzaga and Cordeiro, who demonstrated its suitability to process complex data by answering queries with the concept of “for all”, as follows.

2.2 Similarity-Aware Division

The Similarity-Aware Division [3] is defined as $T_1 [L_1 \div L_2] T_2 = T_R$ where T_1 , T_2 and T_R are relations corresponding to the dividend, the divisor, and the quotient. L_1 and L_2 are lists of attributes of T_1 and T_2 respectively. Thus, both lists must have the same number of attribute and the pair of attribute to be compared must be in the same domain. The quotient relation T_R is defined as the subset of T_1 with the largest possible cardinality, that is, T_1 is partitioned into $\kappa \geq 0$ distinct groups of tuples so that each group $T_{G_k} \subseteq T_1$ represents one candidate tuple for T_R and the following conditions applies: $T_R \times T_2 \subseteq T_1$ and $T_1 = \bigcup_{k=1}^{\kappa} T_{G_k}$. During the division operation, each attribute in L_1 is compared against its counterpart in L_2 .

To illustrate the division, let us consider our toy example in Fig. 1 that illustrates pictures of screw-threads obtained in the factory’s production line. In this setting, relations **ScrewThreads** and **StandardSample** are, respectively, the dividend and the divisor of this operation. The division between these relations will produce a third relation **Approved**, containing the pieces that are similar to the “standard sample”, a flawless screw-thread. The operator will divide relation **ScrewThreads** into two distinct groups by the attribute **ID** based on the active domain, this grouping process is called “**intra-relation comparison**”, as illustrated in Fig. 1 from Introduction, where different groups are represented in different shades of gray. However, If the elements are of complex type, they will be grouped by appropriate similarity grouping algorithm following the problem’s semantic. Then, the groups will be compared according to the similarity of the attribute **Picture** in a process called “**inter-relation comparison**” by means of a distance function and maximum similarity threshold. Finally, the division operator verifies which groups have tuples similar to all tuples in relation **StandardSample** and the groups that satisfy this condition is outputted in relation **Approved**.

To perform the similarity-aware division there are two proposed algorithms [3], one is executed when the attributes in L_1 are indexed and another algorithm based on full table scan when the attributes in L_1 are not indexed. For the remainder of this article, we will refer to their indexed algorithm as **Indexed Algorithm** and the other algorithm as **Full Table Scan**. We also refer to the

relation that identifies each group of the dividend as **grouping relation**. The authors [3] considered the grouping process a separated task since grouping elements by similarity is a research area on its own and the appropriate algorithm depends on the semantics of the problem as well as the data type. Therefore, the grouping relation is a parameter. Both algorithms receive as parameters the divisor relation, the distance function and threshold for each column attribute present in the divisor relation and a relation that identifies the groups that each tuple from the dividend relation belongs to. The **Indexed Algorithm** receives the path to the dividend's index and while the **Full Table Scan** algorithm receives the dividend relation fully. Considering that the divisor relation is generally small, the **Full Table Scan** algorithm builds an in-memory index over that relation to speed up querying process. The idea of both algorithms is to query through one of the indexes looking for groups with similar tuples to the query center. The groups that are present in all resulting queries are groups that meet all the requirements and their ID is outputted in resulting table.

3 Related Work

This section presents the related work on relational division and the inclusion of similarity operators in RDBMS. To illustrate the relational division in SQL, statements 1 to 3 represents the query “*What candidates have all the requirements for the job?*”, using relations `CandidateSkills(CandidateID, Skill)` and `JobRequirements(Skill)`.

3.1 Algorithms for Relational Division

Matos et al. [9] and Gonzaga et al. [10] compared many division statements in SQL and concluded that the counting approach was the fastest statement in every case tested. In a nutshell, as illustrated in Statement 1, the approach counts if the number of matched tuples between a group in the dividend and the divisor is the same as the divisor's cardinality, if it is, the group attends to all division's requirements. The second most efficient approach presented in the work of [10] was the **Subset** and the third most efficient was the **Double Negation**. The **Subset** selects candidates whose set of ability is a superset of the required abilities and the **Double Negation** selects candidates that do not contain a missing requirement. These approaches are represented in Statements 2 and 3, respectively.

<pre>(1) SELECT CandidateID FROM CandidateSkills cs CROSS JOIN JobRequirements jr WHERE cs.Skill = jr.Skill GROUP BY CandidateID HAVING COUNT(*) = (SELECT COUNT(*) FROM JobRequirements);</pre>	<pre> (2) SELECT candidates FROM (SELECT DISTINCT CandidateID candidates FROM CandidateSkills) WHERE NOT EXISTS((SELECT Skill FROM JobRequirements) EXCEPT (SELECT Skill FROM CandidateSkills cs WHERE candidates=cs.CandidateID));</pre>
---	--

Draken et al. [11] proposed a new SQL expression to support Relational Division. The authors included a novelty keyword `DIVIDE` to which users can express division easily on their queries. The query using the new keyword is rewritten into equivalent SQL queries such as the counting approach or the `Double Negation` approach.

```
(3) SELECT DISTINCT CandidateID FROM CandidateSkills csGlobal
WHERE NOT EXISTS( SELECT Skill FROM JobRequeriments jr
WHERE NOT EXISTS( SELECT * FROM CandidateSkills csLocal
WHERE csLocal.CandidateID = csGlobal.CandidateID
AND csLocal.SKill = jr.Skill ) ) ;
```

3.2 Similarity-Awareness in RDBMS

Similarity comparison is not naturally supported by commercial RDBMS, therefore, many researchers have been proposing means to include this type of comparison in RDBMSs. The proposals to include similarity awareness in RDBMSs vary from extending the core of the system to implement external resources that execute similarity-aware operations and return information to the core.

PostgreSQL-IE [12] and **Kiara** [13] encapsulate similarity query algorithms, such as the range and k-nn selection and join, in User Defined Functions in PostgreSQL, **SimDB** [14] is an extension of PostgreSQL's core that implements similarity group by and join algorithms as physical operators.

SIREN [15], **SimbA** [16] are similarity retrieval middlewares embedded between the application and the database that provides an extended SQL to support similarity queries. This SQL allow the representation of similarity-based predicate along with traditional predicates by extending the grammar of standard SQL such as in the `SELECT` command and the `WHERE` clause. As well as extending DDL commands to associate complex data with feature extractors and distance functions. When a query written in this SQL is submitted by the application to the database, the middlewares intercept the query, perform the similarity operators externally and rewrite the query using standard SQL including the tuples outputed from the algorithms in a `WHERE IN` clause. Then the database finishes processing relational operators, if there is any. For instance, Statement 4 performs a range query on the tuples of table `TABLE1` evaluating if their attribute `SIGNATURE` is similar the homonym attribute of the element 1 in `TABLE2` up, to a threshold of 2. The middleware will process the query and rewrite it as in Statement 5 and submit to the database system. Note that the second predicate is not processed by the middleware. All standard predicates are processed by the RDBMS. **SimbA** is an extension of **SIREN** that allows the inclusion of distance functions and feature extractos dynamically.

```
(4) SELECT * FROM | (5) SELECT *
FROM TABLE1 | FROM TABLE1
WHERE SIGNATURE | WHERE ID IN (1,2,3,6,9..)
NEAR (SELECT SIGNATURE FROM TABLE2 | AND att2 = 'landscape';
WHERE ID = 1) RANGE 2 USING euclidean |
AND att2 = 'landscape'; |
```

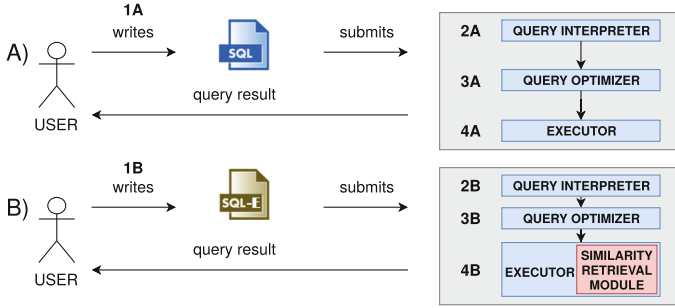


Fig. 2. (a) Sequence to execute relational division in RDBMS. (b) Our proposed sequence to execute the similarity-aware division.

Finally, **FMI-SiR** [17] is a module that uses interface and resource extensions provided by Oracle to implement DBMS-linked data types, feature extractors, distance functions and indexes for similarity-based retrieval. The module is implemented as a C++ Oracle library and defines methods in SQL that maps to C++ functions, such as the one in Statement 6. Then, the function can be used in any SQL query as part of the WHERE clause, such as in the statement in 7 that executes the same range query illustrated in Statement 4.

```
(6) CREATE FUNCTION                               |(7) SELECT *
euclidean_distance                               |FROM TABLE1
(signature BLOB, signature2 BLOB)                |WHERE euclidean_dist(signature,
RETURN float AS                                 |(SELECT signature FROM
LANGUAGE C LIBRARY libfmisir                    |TABLE2 WHERE id = 1)) <= 2
NAME "euclideanDistance"                        |att2 = 'landscape';
WITH CONTEXT;                                   |
```

During query execution, the RDBMS changes context from SQL to C++, executes the similarity algorithm and returns the result to the RDBMS which continues to process the query. The query processor can manipulate the invocation anywhere in the query plan, allowing query optimization.

4 Including the Similarity-Aware Division Operator in a Commercial RDBMS

This section details our proposed inclusion of the similarity-aware division operator on Oracle RDBMS with FMI-SiR library attached to it.

Figure 2a illustrates the RDBMS’s architecture to execute the original Relational Division. In Step 1a, the user writes the query in SQL using a Statement such as 1, 2 or 3. In Step 2a, the query is submitted to the query interpreter for parsing. Step 3a optimizes the query based on relational algebra’s rules and on access cost and data’s statistics. This optimized plan is executed in Step 4a. Finally, the query result is outputted to the user in Step 5a.

We propose to do the same for the Similarity-Aware Division. As illustrated in Fig. 2b, first we propose intuitive syntaxes to represent this operator using an SQL with support to similarity operators (Step 1b). Then, we map these syntaxes to a SQL executable by the RDBMS (Step 2b). We study these syntaxes' runtime execution inside the RDBMS and obtain hints to when each syntax is the fastest. These hints can be used during the query optimization (Step 3b). Finally, an improved query is rewritten and executed by the RDBMS (Step 4b) using a similarity retrieval module. The query result is outputted to the user in Step 5b.

4.1 The Proposed Syntax for Similarity-Aware Division

We studied the division's SQL statements from the works of Matos et al. [9] and Gonzaga et al. [10] and carefully designed strategies that allowed us to create similarity-aware versions for them.

Experimental evaluations in their work show that the counting approach (Statement 1) is the most efficient to execute the original relational division. Unfortunately, this approach cannot be efficiently adapted to resolve similarity division. As illustrated in Statement 1, for each group in the dividend, this approach counts how many tuples are equal to the tuples in the divisor. By equality, each tuple in the group matches exactly one tuple in the divisor. However, by similarity, a single tuple in a group can be similar to multiple tuples in the divisor, i.e., a single tuple can match all the tuples in the divisor and thus qualify its group or two distinct tuples in a group can be similar to the same tuple in the divisor. In these scenarios, the number of tuples that are similar to the tuples in the divisor is uneven for each group. Some groups can have more tuples, fewer tuples or equal tuples and still qualify for the operation.

The similarity operators can be represented in SQL via user defined functions, such as in Statement 7, or by extension of the SQL's syntax, such as in Statement 4. We choose the latter approach to represent the Similarity-Aware Division operator, in Step 1b, as it allows the manipulation of complex data in a similar manner to regular data types, such as number and characters. The execution of these operations is implicit to the user. In this regard, we employed **SIREN**'s similarity-oriented SQL. To include similarity comparison in our designed adaptations, we changed the comparisons originally performed by equality in the statements. The intra-relation comparisons are now represented by means of a distance function and a threshold. The inter-relation comparison is provided by joining the dividend relation with its grouping relation and projecting the feature vectors along with the group identifier instead of the primary keys from the original relation.

The **Subset** approach requires a similarity difference operator. To represent this operator we added a new syntax rule to the SQL, illustrated in Statement 8. This novel syntax rule follows the same principles of other similarity operators in the employed extended SQL [15]. The phrase **RANGE EXCEPT** expresses the similarity difference operator, the keyword **NEAR** provides the similarity threshold for each attribute participating in the operation and the keyword **USING** specifies the respective distance functions.

The Statements 8 and 9 adapts the **Subset** and the **Double Negation** approaches to perform similarity division and answer the query of our toy dataset in Fig. 1: “*What products have all the requirements of the industrial quality control?*”. In both statements relation **ScrewThreads** is the dividend, relation **StandardSample** is the divisor and **SThreadGroups** is the resulting table from an grouping algorithm that relates each tuple from **ScrewThreads** to their respective group and the attribute ID is replaced by **groupID**.

```
(8) SELECT groupIDs FROM                               |(9) SELECT DISTINCT groupID
(SELECT DISTINCT groupID                               |FROM SThreadGroups ST1
  groupIDs FROM SThreadGroups)                       |WHERE NOT EXISTS(
WHERE NOT EXISTS(                                     |  SELECT Pictures FROM
  (SELECT Pictures FROM StandardSample)               |  StandardSample S
  RANGE EXCEPT                                     |  WHERE NOT EXISTS(
  (SELECT Pictures FROM ScrewThreads                 |    SELECT * FROM ScrewThreads
    JOIN SThreadGroups ON                             |    JOIN SThreadGroups ON
    ID = ElementID stg WHERE                         |    ID = ElementID ST2 WHERE
    stg.groupID = groupIDs)                           |    ST1.groupID = ST2.groupID
  ) NEAR 1 USING euclidean));                          |    AND ST1.Picture RANGE
                                                         |    S.Picture NEAR 1 USING
                                                         |    euclidean));
```

4.2 Algorithms to Perform the Similarity-Aware Division

To execute the similarity-aware division the statements 8 and 9 are rewritten to the supported SQL in Oracle where similarity algorithms are represented in user-defined functions, as in Step 2b. We choose this module for similarity retrieval because it delegates the execution of this operators to the RDBMS query processor, allowing optimization by query rewriting. Statements 10 and 11 represent the rewriting of Statements 8 and 9.

```
(10) SELECT groupIDs FROM                               |(11) SELECT DISTINCT groupID
(SELECT DISTINCT GroupID                               |FROM SThreadGroups S
  groupIDs FROM SThreadGroups)                       |WHERE NOT EXISTS(
WHERE NOT EXISTS(                                     |  SELECT Pictures FROM
  SELECT * FROM TABLE(SIMEXCEPT(                  |  StandardSample S2
  CURSOR(SELECT Pictures FROM                          |  WHERE NOT EXISTS(
  StandardSample),CURSOR(SELECT                       |    SELECT * FROM ScrewThreads
  Pictures FROM ScrewThreads JOIN                     |    JOIN SThreadGroups ON
  SThreadGroups ON ID = ElementID                    |    ID = ElementID ST2
  stg WHERE stg.groupID = groupIDs),                 |    WHERE S.groupID = S2.groupID
  'euclidean', '1'))                                 |    AND euclidean_dist(S.Picture,
);                                                    |    S2.Picture) <= 1));
```

The **Double Negation** syntax in Statement 11 may employ a metric access method to speed the operation. The **Subset** syntax, in Statement 10, requires a similarity difference algorithm. Though there are algorithms proposed in the literature, they have limitations such as only working for unidimensional data

or not allowing tuples from the same relation to be similar to each other. Thus we propose a similarity difference algorithm named **SIMEXCEPT** that works as it follows: Given two relations **R** and **S**, our novel algorithm iteratively builds an in-memory index for each attribute of **S**. The elements from the **R** relation are used as the query center. If an element appears in every index result from the same query object, the query object is excluded from the answer, i.e., there is a similar element to that query object.

As well as extending well-known SQL division statements, we implemented the algorithms described in Sect. 2.2 as table functions on FMI-SiR through Oracle's interface **ODCITable**. This implementation allows the algorithms to be invoked in an SQL statement such as in Statement 12, that invokes the **Full Table Scan** algorithm. This approach is the same used to implement the similarity difference algorithm from Statement 11. In both cases, the first parameters are tables resulted from queries - this approach allows the operator to be invoked anywhere in the query plan - and the last two indicate the distance functions and the similarity threshold.

```
(12) SELECT * FROM TABLE(FTSDivide(CURSOR(SELECT * FROM ScrewThreads),
CURSOR(SELECT * FROM StandardSample),CURSOR(SELECT * FROM SThreadSamples),
'euclidean', '1'));
```

5 Experiments and Discussion

We created synthetic datasets that represent possible input relations to perform similarity division. Remember that the division is represented as $T_1 [L_1 \div L_2] T_2 = T_R$. Therefore, in our experiments, the dividend (T_1) is defined as $T_1(\text{ID}, \text{SIGNATURE})$ and the divisor (T_2) is defined as $T_2(\text{SIGNATURE})$. For each relation, respectively, L_1 and L_2 contain the attribute **SIGNATURE** that is a five-dimensional array with values varying from 0 to 1 stored in the RDBMS as a BLOB. The experiments were to evaluate the scalability of the algorithms in an RDBMS environment. We tested the algorithms in an Intel 2 machine with CPU of 2,67 GHz, 32 GB of RAM running Ubuntu. The Oracle version used was the 12c with the FMI-SiR library installed.

Unless stated otherwise, the cardinality of the dividend is 100 thousand tuples while the cardinality of the divisor is 5 tuples. We did not create large divisor relations because division with too many requirements is likely to be useless, returning a very limited or empty result set. A huge number of tuples in the divisor makes the division operation too restrictive as it requires groups to have tuples similar to all of them. The total number of groups is fixed to 10 but only one group is considered valid, that is, meets all the requirements. The tuples were inserted on the group in an exponential fashion, that is, the higher the group's id, the more tuples it has. In the graphics and the following paragraph **Subset** refers to statement 10, **Double Negation** refers to Statement 11, the **Indexed Double Negation** refers to the same statement, however, access is done via index. **Indexed Algorithm** refers to the implementation of the algorithm that uses a pre-built index - the same used in the **Indexed Double Negation** -

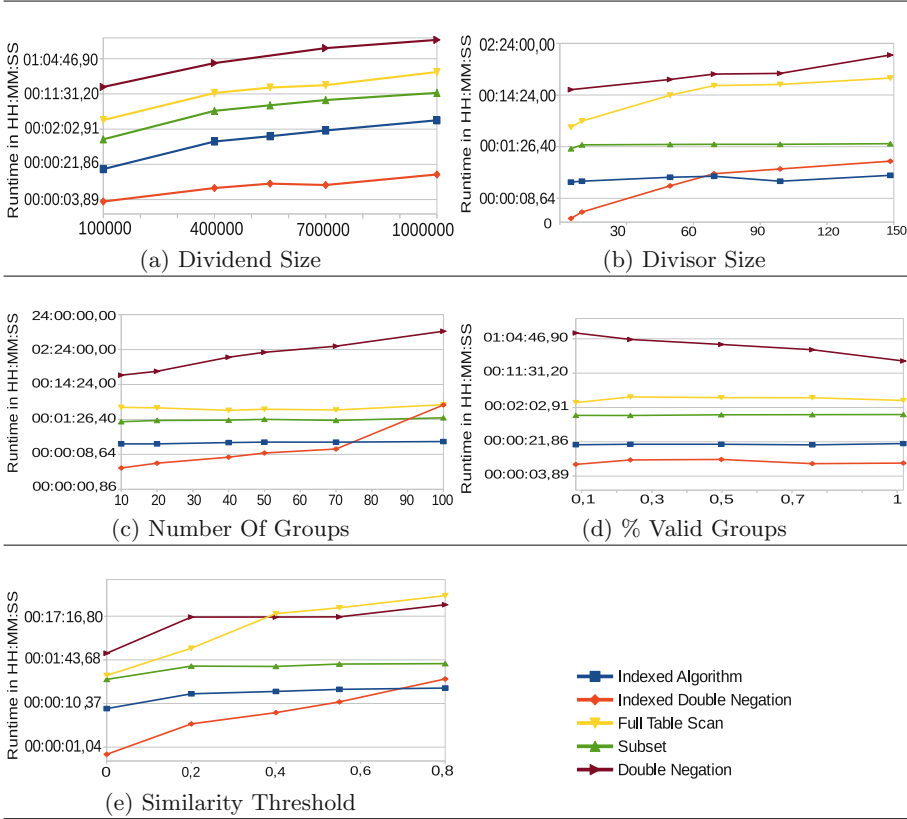


Fig. 3. Runtime of similarity-aware division algorithms. Consider the operation represented as $T_1 [L_1 \div L_2] T_2 = T_R$. a-is the runtime when increasing the number of tuples of the **dividend** (T_1). b-is the runtime when increasing the number of tuples of the **divisor** (T_2). c-is the runtime when increasing the **number of groups** in the dividend relation. d-is the runtime when increasing the number of tuples of the **quotient/valid groups** (T_R). e-is the runtime when increasing the **similarity threshold between the attributes in L_1 and L_2** , that is attribute in the dividend that is compared with the attribute in the divisor.

and **Full Table Scan** refers to the implementation of the algorithm that reads the dividend table fully. The y-axis is always presented in log scale.

Each of these alternatives perform the similarity division operation in different manners. The **Subset** and the **Double Negation** (Statements 10 and 11, respectively) are SQL statements containing a set of operations and/or sub-queries to which the RDBMS has freedom to choose the best execution plan while the **Indexed Algorithm** and **Full Table Scan** algorithm are the implementations of the two state-of-art algorithms describe 2.2 in section executed by the DMBS through statements similar to Statement 12.

The **Indexed Double Negation** is a variant of the **Double Negation** query that relies on index to speed up execution. It is interesting to notice that, thanks to the index, the former variant presented very satisfiable results in almost every experiment, the latter presented the worst execution time in nearly every case, with the sole exception when increasing the similarity threshold.

Figure 3a shows the run time of the algorithms when increasing the dividend size. The **Indexed Double Negation**, was the fastest in every case tested. This is because of the inter-table comparison task. While **Indexed Algorithm** and **Full Table Scan** algorithm load the entire grouping relation in memory, this algorithm only joins the query result elements with a filtered grouping relation. The **Full Table Scan** and the **Subset** are the reverse versions of each other. While the former builds an index over divisor table, the later builds the index over the dividend table but both load the dividend relation in memory, hence they are so slow.

Figure 3b shows the runtime of the algorithm when increasing the divisor size. The divisor size varies from 5 to 100 tuples. The **Indexed Double Negation** algorithm presented an increasing curve, implying that the number of tuples in the divisor relation is directly related to this algorithm's performance. High values of divisor tuples also affected the **Full Table Scan** algorithm as it requires more accesses to a small index. The other algorithms did not show much impact regarding this parameter.

Figure 3c shows the run time of the algorithms when increasing the total number of groups, varying from 10 groups to 100. The **Indexed Double Negation** showed to be more sensitive to this parameter as increasing the number of groups also increases the number of calls to the intern query where the index is accessed resulting in more disk access.

Figure 3d shows the run time of the algorithms when increasing the number of valid groups, i.e., groups that satisfy the requirements. The total number of groups was fixed to 40. The x-axis represents the percentage of valid groups where 0.01 is 5 groups and 1 is 40 groups. This parameter did not seem to affect the algorithms.

Figure 3e shows the run time of the algorithms when increasing the similarity threshold. The value represents the percentage of the dataset's estimated diameter, in which 0 means identity comparison and 1 means all elements are similar to each other. This parameter did not seem to affect the **Indexed Algorithm** and **Subset**, however, the **Full Table Scan** and **Indexed Double Negation** showed to be more sensitive to large threshold values. This is due to a well-known problem regarding indexes in general, provided that they lose performance in queries with low selectivity. The **Full Table Scan** algorithm has an index over the divisor table, the index is degrading fast to a linked-list. As for **Indexed Double Negation**, while still using the very same index of **Indexed Algorithm** it differs in the access form and the memory space used to retrieve the elements.

Discussion: As shown in Fig. 3, if there are no existing indexes for the complex attributes in the dividend relation that are compared with the complex attributes in the divisor relation, the **Subset** approach is more efficient than the **Full**

Table Scan algorithm in all experimented cases. If there are indexes there are cases when the **Index Double Negation** approach is more efficient than the **Indexed Algorithm**. Thus, during the rewriting step from the extended SQL to an Oracle-executable SQL - discussed in Sect. 4.2 - the analyzer must check divisor's size and the number of groups. For increasing values in the divisor or in the number of groups, regardless of how the SQL statement is written, it must be rewritten to perform the **Indexed Algorithm**. However, on smaller values, it must be rewritten to perform the **Indexed Double Negation** approach.

6 Conclusion

In this article, we propose a new SQL syntax for the similarity-aware division operator based on the relational division syntax and an similarity-oriented SQL. We then studied their implementations and compared them with two state-of-art algorithms to perform the similarity-aware division operation on a commercial database, namely Oracle. After comparing each implementation in order to find bottlenecks in many different situations, we identified hints to when one algorithm is better than the other. We found that if there is an existing index for the attribute in the dividend that is compared with an attribute in the divisor relation, the best approach is to perform similarity division on a small number of divisor elements or groups is the **Indexed Double Negation** while for greater values the **Indexed Algorithm** should be used. When there is no index, the **Subset** is the most appropriated algorithm to be used.

Future works include extending the experiments to real datasets and using the hints to rewrite the similarity-aware division operator query to the appropriated statement accordingly to the evaluated parameters, providing a more efficient way to perform the operation, i.e., if one write a query using the **Indexed Double Negation** and the divisor is too small, the query analyzer rewrites the query to perform the operation using the **Indexed Algorithm**. Considering the complexity to write these queries, we find interesting to evaluate the necessity of a new keyword, to represent the operation in a more intuitive manner similar to the work of [11].

Acknowledgements. We would like to thank CNPq, CAPES project 10357907/M and FAPESP project 2016/170780 for financial support.

References

1. Codd, E.F.: The Relational Model for Database Management, Version 2. Addison-Wesley, Boston (1990)
2. Vasconcelos, G.Q., et al.: Tender-sims - similarity retrieval system for public tenders. In: ICEIS 2018, pp. 143–150 (2018)
3. Gonzaga, A.S., Cordeiro, R.L.F.: A new division operator to handle complex objects in very large relational datasets. In: EDBT 2017, pp. 474–477 (2017)
4. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. *ACM Comput. Surv.* **33**(3), 273–321 (2001)

5. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach, 1st edn. Springer, Heidelberg (2010). <https://doi.org/10.1007/0-387-29151-2>
6. Marri, W.J.A., Malluhi, Q.M., Ouzzani, M., Tang, M., Aref, W.G.: The similarity-aware relational database set operators. *Inf. Syst.* **59**, 79–93 (2016)
7. Pola, I.R.V., Cordeiro, R.L.F., Traina Jr., C., Traina, A.J.M.: Similarity sets: a new concept of sets to seamlessly handle similarity in database management systems. *Inf. Syst.* **52**, 130–148 (2015)
8. Silva, Y.N., Aref, W.G., Ali, M.H.: Similarity group-by. In: Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, 29 March 2009–2 April 2009, Shanghai, China, pp. 904–915 (2009)
9. Matos, V.M., Grasser, R.: Assessing performance of the relational division operator. In: Database Management. Auerbach Publications, February 2001
10. Gonzaga, A.S., Cordeiro, R.L.F.: Fast and scalable relational division on database systems. In: SBBD 2016, pp. 169–174 (2016)
11. Draken, E., Gao, S., Alhajj, R.: Making query coding in SQL easier by implementing the SQL divide keyword: an experimental query rewriter in Java. In: Advanced Database Query Systems: Techniques, Applications and Technologies, 1st edn. IGI Global (2001)
12. Guliato, D., Melo, E.V., Rangayyan, R.M., Soares, R.C.: POSTGRESQL-IE: an image-handling extension for PostgreSQL. *J. Digit. Imaging* **22**(2), 149–165 (2009)
13. Oliveira, P.H., et al.: On the support of a similarity-enabled relational database management system in civilian crisis situations. In: ICEIS 2016, pp. 119–126 (2016)
14. Silva, Y.N., Aly, A.M., Aref, W.G., Larson, P.: SimDB: a similarity-aware database system. In: SIGMOD 2010, pp. 1243–1246 (2010)
15. Barioni, M.C.N., Razente, H., Traina, A., Traina Jr., C.: SIREN: a similarity retrieval engine for complex data. In: VLDB 2006, pp. 1155–1158 (2006)
16. Bedo, M.V.N., Traina, A.J.M., Traina Jr., C.: Seamless integration of distance functions and feature vectors for similarity-queries processing. *JIDM* **5**(3), 308–320 (2014)
17. dos Kaster, D.S., Bugatti, P.H., Traina, A.J.M., Traina Jr., C.: FMI-SiR: a flexible and efficient module for similarity searching on Oracle database. *JIDM* **1**(2), 229–244 (2010)

Data Quality and Data Cleansing



Data Quality in a Big Data Context

Franco Arolfo  and Alejandro Vaisman 

Instituto Tecnológico de Buenos Aires, Buenos Aires, Argentina
{farolfo, avaisman}@itba.edu.ar

Abstract. In each of the phases of a Big Data analysis process, data quality (DQ) plays a key role. Given the particular characteristics of the data at hand, the traditional DQ methods used for relational databases, based on quality dimensions and metrics, must be adapted and extended, in order to capture the new characteristics that Big Data introduces. This paper dives into this problem, re-defining the DQ dimensions and metrics for a Big Data scenario, where data may arrive, for example, as unstructured documents in real time. This general scenario is instantiated to study the concrete case of Twitter feeds. Further, the paper also describes the implementation of a system that acquires tweets in real time, and computes the quality of each tweet, applying the quality metrics that are defined formally in the paper. The implementation includes a web user interface that allows filtering the tweets for example by keywords, and visualizing the quality of a data stream in many different ways. Experiments are performed and their results discussed.

Keywords: Data quality · Social networks · Big Data

1 Introduction and Motivation

The relevance of so-called Big Data has been acknowledged by researchers and practitioners even before the concept became widely popular through media coverage [1]. Although there is no precise and formal definition, it is accepted that Big Data refers to huge volumes of heterogeneous data that must be ingested at a speed that cannot be handled by traditional database systems tools. Big Data is characterized by the well-known “4 V’s” (volume, variety, velocity, and veracity), implying that not only the data volume is relevant, but also the different kinds of structured, semistructured and unstructured data, the speed at which data arrives (e.g., real time, near real time), and the reliability and usefulness of such data. However, it is also acknowledged that most of the promises and potential of Big Data are far from being realized. This gap between promise and reality is due to the many technical problems and challenges that are usually overlooked, although the database research community has warned about them, namely heterogeneity, scale, timeliness, complexity, and privacy, among other ones [3].

In each of the phases in a Big Data scenario, data quality (DQ) plays a key role, as the very nature of the “4 V’s” suggest. The largely studied concepts of DQ must be revisited and re-studied in a Big Data context, since, as it will be discussed in this paper, many new problems appear, which are not present

in traditional relational databases scenarios [8]. Intuitively, each of the “V’s” define a different context for data analysis, and therefore, for DQ. Thus, there is a strong relationship between the work about contexts in DQ (e.g., [7, 11]) and the problems of DQ in Big Data, since different notions of quality must be used for different types of Big Data. In particular, this paper deals with DQ in a real-time scenario, specifically Twitter feeds. This is a typical scenario where data come at high speed, highly unstructured, and with very volatile reliability and usefulness. All of these characteristics are the complete opposite of a relational database analytics scenario, where data are highly structured, and cleaned, transformed and analyzed offline. Therefore, DQ must be addressed considering these differences.

In spite of the relevance of the topic, there has been not much work so far, in particular regarding the implementation of quality processes over Big Data sources. This paper tackles this issue. More concretely, the contributions of this work are: (1) The definition of DQ dimensions and metrics in a Big Data scenario where data arrive as unstructured documents and in real time. Traditional DQ dimensions are redefined, to address those particular characteristics. This general scenario is instantiated to study the concrete case of Twitter feeds. (2) The implementation of a system that acquires tweets in real time, and computes the quality of each tweet, applying the quality metrics defined formally in the paper. The implementation includes a web user interface that allows filtering the tweets e.g., by keywords, computing their data quality, and visualizing the DQ, not only the overall one, but also along each dimension. (3) An experimental study of the quality of the feeds, using the tool described above. This study is aimed at showing how DQ can be used to determine the attributes that characterize the different quality of the tweets, filter out bad quality data, or validate the conclusions drawn in the data analysis phase.

The remainder of the paper is structured as follows. Section 2 discusses related work. In Sect. 3, the traditional DQ dimensions and metrics are presented, while Sect. 4 studies the DQ dimensions and metrics for Big Data. Section 5 presents the computation of DQ metrics to evaluate the quality of a tweet based on a collection of weighted metrics. Section 6 introduces the implementation of the system, and Sect. 7 presents an experimentation and reports and discusses the results. Section 8 concludes the paper.

2 Related Work

Ensuring the quality of data in databases has long been a research topic in the database community. Research in DQ has resulted in the definition of dimensions, metrics, and methods to assess the quality of a database [16], and also in an ISO standard specification¹. In spite of this, classic research considers DQ as a concept independent of the context in which data are produced and used, which is clearly not enough to solve complex problems, particularly in current times, when, among other facts, ubiquitous computing requires accounting for space and time when a query is being answered. Strong et al. [14] realized this problem, and claimed that data quality is highly dependent on the context,

¹ <http://iso25000.com/index.php/en/iso-25000-standards/iso-25012>.

which became an accepted fact thereon. The rationale for this conclusion was based on the fact that, similarly to quality in general, DQ cannot be assessed independently of the consumers who choose and use the products (e.g., [11]). The former proposes a system where contextual information allows evaluating the quality of blood pressure data. The latter proposes a framework that allows context-sensitive assessment of DQ, through the selection of dimensions for each particular decision-maker context and her information requirements.

There is a large corpus of work regarding data context management with a wide variety of uses. It is widely accepted that most modern applications, particularly over the web, are required to be context-aware. Bolchini et al. [6] presented a survey of context models, with a well-defined structure, that identified some important aspects of context models. In particular, they remark that models must account for *space*, *time*, *context history*, *subject*, and *user profile*. Preferences in databases have also been extensively studied [7, 13]. In the multi-dimensional databases domain, [10] proposes to define the context through the use of logic rules, representing the database as a first-order logic theory.

Recently, [4, 8] study the particularities of data quality in the context of Big data, that is, how the “4 V’s” mentioned in Sect. 1 impact on well-known DQ dimensions and metrics used in traditional structured databases [5]. The main message in [8] is that Big Data quality should be defined in source-specific terms and according to the specific dimension(s) under investigation. In some sense, this means that the context is again present in the Big Data scenario when quality is addressed. The present paper builds from these studies, as will be clear in the remaining sections.

3 A Short Background

Data Quality (DQ) is a multi-faceted concept, represented by different dimensions, each one referring to a different quality aspect [5, 14]. A *DQ dimension* captures a facet of DQ, while a *DQ metric* is a quantifiable instrument that defines the way in which a dimension is measured. Since a DQ dimension is in general a wide concept, an associated metric allows specifying a concrete meaning for the dimension. As a consequence, many different metrics can be associated to the same DQ dimension, and their application will measure several different aspects of the dimension. In a broader sense, the *quality of an object or service* represents how much this object or service fits the needs to solve a given problem. That is, quality is not absolute to the object or service *per se*, but relative to the problem to be solved. This is the approach followed in this work.

3.1 Dimensions and Metrics

While a large number of DQ dimensions were proposed in the literature, there is a basic set of them, which are generally acknowledged to be representative of the quality of data [5, 12]. This set includes accuracy, completeness, consistency, freshness (or timeliness), among other ones.

- *Accuracy*: Specifies how accurate data are, and involves the concepts of *semantic accuracy* and *syntactic accuracy*. The former refers to how close is

a real-world value to its representation in the database. The latter indicates if a value belongs to a valid domain. In other words, it describes the closeness between a value v and a value v' , considered as the correct representation of the real-life phenomenon that v aims at representing. For example, if someone wants to type the name “John” but typed “Jhn”, there is an accuracy issue.

- *Completeness*: Represents the extent to which data are of sufficient breadth, depth, and scope for the task at hand. For relational databases, this can be characterized as the presence/absence and meaning of null values, assuming that the schema is complete.
- *Redundancy*: Refers to the representation an aspect of the world with the minimal use of information resources.
- *Consistency*: Refers to the capability of the information to comply without contradictions with all the rules defined in a system. For example, in a relational database constraints are defined to guarantee consistency.
- *Readability*: Refers to the ease of understanding of information. This could be the case when, for example, a hand-written paragraph is scanned, and some of the characters are not well defined.
- *Accessibility*: Also called *availability*, is related to the ability of the user to access the information.
- *Trust*: Refers to how much the information source can be trusted, and therefore to what extent data are reliable. For example, people may rely on Facebook or Twitter posts to find out the quality of a movie, or check the IMDB site at <http://www.imdb.com>, which might provide more reliable data.
- *Usefulness* (cf. [8]): This is related to the benefits a user can obtain when using the data to produce information. For example, to observe technical details present in a picture of a painting, a user would choose the image with the highest contrast. Again, this is also a contextual quality dimension: a lower-quality picture may be enough for some users or for some kinds of requirements, while clearly not enough when the details are needed.

To quantify these dimensions and to be able to assess DQ according to them, the concept of *metrics* must be introduced. Mathematically, a DQ *metric* for a dimension D is a function that maps an entity to a value, such that this value, typically between 0 and 1, indicates the quality of a piece of data regarding the dimension D . For a given dimension, more than one metric could be defined and combined to obtain a concrete quality value. Note that metrics are highly context-dependent. For example, the readability of a hand-written text may be influenced not only by the text content, but also by the way the user writes. The same occurs with metrics for other DQ dimensions.

3.2 Big Data Quality

In a Big Data context, datasets are too large to store, analyze, handle or process, for the traditional database tools. As explained above, Big Data are characterized by the well-known “4 V’s”, namely *Volume* (size of the datasets), *Velocity* (speed of incoming data, e.g., the number of tweets per second (TPS)), *Variety* (refers to the type and nature of the data), and *Veracity* (the reliability of the data, which, in this context, is greatly volatile, even within the same data

stream). In the literature, many other “V’s” can be found, but only these four will be considered in the present work. According to the structure of data, they can be classified in: (a) *Structured*, where each piece of information has an associated fixed and formal structure, like in traditional relational databases; (b) *Semi Structured*, where the structure of the data has some degree of flexibility (e.g., an XML file with no associated schema, or a JSON response from an API, whose structure is not completely defined); (c) *Unstructured*, where no specific structure is defined. Further, the United Nations Economic Commission for Europe (UNECE) classifies Big Data according to the data sources in: *Human sourced*; *Process mediated*; and *Machine generated* [15]. These are explained next.

- *Human-sourced data*: Information people provide via text, photos or videos. Usually, this information lacks of a fixed structure, like the texts written in natural language. Therefore, the information streamed here is *loosely structured* and often ungoverned. Examples are social networks posts (Facebook, Twitter), YouTube videos or e-mails, and, in general, data coming from social networks.
- *Process-mediated data*: This is the information that concerns some business events of interest, like the purchase of a camera in an e-commerce site or the sign-up of clients in a system. This information is *highly structured*, such as relational databases, coming from traditional Business systems.
- *Machine-generated data*: Refers to the data resulting of the tracking of sensors of the physical world (e.g., temperature sensors, human health sensors, GPS coordinates, etc.). This type of source is associated with very large amounts of data, given the constant tracking of the sensors. In general, these are data coming from the so-called Internet of Things.

Given these characteristics of Big Data, the DQ along the dimensions explained in Sect. 3.1 must be quantified using metrics specific to such a context, therefore the typical quality metrics used for structured, process-mediated data must be adapted to this new situation. This is studied in the next section.

4 Data Quality Dimensions and Metrics in a Big Data Context

This section studies how the DQ dimensions can be used in a Big Data scenario. The study focuses in *human-sourced generated data*. The next sections describe how these dimensions can be applied to address the quality of Twitter² streams. Metrics for the dimensions defined here are presented later.

- *Readability (r)*. Given a dictionary D , and a collection of words considered as valid in a document x , the *Readability* of x , denoted $r(x)$ is defined as the quotient between the valid words in x and all the words in x , if any, otherwise it is zero. That is, given a set W of the words (valid and non-valid) that are present in the document x , the readability of x is

$$r(x) = \begin{cases} \frac{\#\{w \in W \wedge w \in D\}}{\#\{w \in W\}} & \text{if } W \neq \emptyset \\ 0 & \text{if } W = \emptyset \end{cases}$$

² <http://www.twitter.com>.

In the remainder, the problem to be addressed will refer to tweets in a Twitter stream, thus x will represent a tweet.

- *Completeness (c)*. Consider an object x in domain, and an array $props_p$ that contains the names of the properties required to describe x for a given problem p ; assume that x is represented as a collection of (*property, value*) pairs of the form $\{(p_1, v_1), \dots, (p_n, v_n)\}$, such that v_i is a value for p_i . If a property $p_i \in x$ has associated a non-null value v_i , it is called well-defined. There is also a function *validPropsOf* that, given an object x retrieves the set of well-defined properties in it. The *Completeness* of x , denoted $c(x)$ tells if all the properties in $props_p$ are well-defined in x . It is computed as:

$$c(x) = \begin{cases} 1 & \text{if } props_p \subset validPropsOf(x) \\ 0 & \text{otherwise} \end{cases}$$

Example 1. Given an object (a tweet) x , such that $x = \{text: "I like Bitcoin", user: null\}$, and an array $props_p = [text]$, it follows that $c(x) = 1$. Consider now $props_p = [text, user]$. In this case, $c(x) = 0$, since the user property is not well defined, because it has a null value.

- *Usefulness (u)*. Since this paper deals with human-sourced datasets, it will be assumed that this property is directly related to the possibility of (among others):
 - (a) Detecting a sentiment, whether positive or negative, in an object x , say a tweet or post. Therefore, if x reflects a positive or negative feeling about a certain topic or person, x will be considered useful. If the sentiment is neutral, or no sentiment could be computed by a Natural Language Processing (NLP) tool, x will be considered not useful.
 - (b) Detecting the domain or topic of x , for example, politics, marketing, sports, and so on.

Many other ways of assessing usefulness could be considered, but this is outside the scope of this paper. In the remainder, Usefulness is defined as:

$$u(x) = \begin{cases} 1 & \text{if } (sentiment(x) = P \vee sentiment(x) = N) \\ 0 & \text{otherwise} \end{cases}$$

- *Trustworthiness (t)*. In a social network (or, in general, for human-sourced datasets) anyone in general can publish any kind of information anywhere, whether truthful or not. Although this is mentioned here for completeness, validating the trustfulness of a post is outside the scope of this paper.

5 Computing Data Quality

As discussed above, the definition of DQ is not the same for all contexts and problems, but normally it depends on them. That is, DQ depends on the problem and the domain model at hand. This section provides a wide and general definition of DQ, that can be instantiated as needed.

Definition 1 (Problem (p)). A problem p is defined as a string or sequence of characters that defines the problem to be solved in a human-readable way.

Example 2. A problem can be defined as: *Given this Twitter feeds stream, what are the best quality tweets for the hashtag #2020Elections?*

Definition 2 (Domain Model (X)). The set of objects whose quality will be measured.

Example 3. For the case that will be studied in the next section, the domain model is defined as a set of Twitter feeds.

Definition 3 (Data Quality Metric (m_{X_p})). A Data Quality Metric is a function $m_{X_p} : X \rightarrow [0 \dots 1]$, such that, given $x \in X$, and a problem p , then $m_{X_p}(x) = 0$ if x contains data of very poor quality for the given problem p , and $m_{X_p}(x) = 1$ if x contains data of very good quality to fit the problem.

Definition 4 (Weight of a Data Quality Metric ($m_{X_p.weight}$)). Each DQ metric has an associated weight, which is a scalar value between 0 and 1, that measures the relevance of the metric for solving the problem p .

Definition 5 (Data Quality (Q_{X_p})). Consider a problem p , a domain X , and a set of metrics $M_{X_p} = \{m_1, m_2, \dots, m_n\}$. Each m_i is a DQ metric function. Note that n is an integer number greater than zero and the set M_{X_p} is finite. Data Quality (Q_{X_p}) is a function $Q_{X_p} : X \rightarrow [0 \dots 1]$ such that $Q_{X_p}(x) = g(m_1, m_2, \dots, m_n)(x)$, where g is a function $g : (X \rightarrow [0, 1])^n \rightarrow (X \rightarrow [0 \dots 1])$.

In this paper, the quality of a tweet x will be calculated as $Q(x) = g_{(r,c,u)}(x) = \sum_{m=\{r,c,u\}} m(x) * m.weight$, where r , c and u are the metrics for *Readability*, *Completeness* and *Usefulness* respectively, defined in Sect. 4.

Example 4. The Quality value of the following tweet x , using the weights values $r.weight = 0.5$, $c.weight = 0.25$ and $u.weight = 0.25$, is computed as follows.

- text: ‘‘I love Big Data Quality m#a!sc’’
 - id: 1
 - coordinates: [48.864716, 2.349014]

- *Readability* (r)

$$r(x) = \frac{\#\{I, love, Big, Data, Quality\}}{\#\{I, love, Big, Data, Quality, m\#a!sc\}}$$

$$r(x) = \frac{5}{6} = 0.833$$

- *Completeness* (c). Consider that $props_p = \{text, id\}$. Then:

$$\{text, id\} \subset \{text, id, coordinates\}, \text{ and } c(x) = 1.$$

- *Usefulness* (u). The text provided expresses positive sentiment, thus:

$$sentiment(x) = P, \text{ and } u(x) = 1.$$

Finally, the quality value for x is $Q(x) = 0.83 * 0.5 + 1 * 0.25 + 1 * 0.25 = 0.915$.

6 Implementation

This section presents and describes the implementation of the concepts explained in previous sections, applying them to analyze the quality of Twitter feeds streams. The architecture is described first, detailing the technological components and how they interact with each other for capturing, filtering, and displaying the results. Finally, the user interface is described. The goal of the implementation is to develop a system that can let users to analyze a stream of Twitter feeds, based on a particular keyword-led search, and visualize the results to gain insight on the quality of the requested data.

The core of the system is an Apache Kafka³ cluster of three brokers. Kafka is a distributed streaming platform for capturing, processing and storing data streams. The implemented cluster has three nodes running Kafka. A Zookeeper⁴ service coordinates the cluster and manages the message topics. Besides the Kafka core, there are three components, one to produce data, one to consume data, and one to process and display data. Figure 1 illustrates these components and their orchestration. The components are briefly described next.

- *Kafka Producer Service*: A Java 8 program exposing a REST API using the Spring Boot⁵ framework. This API starts searches over the Twitter API and publishes the data to a particular *topic*.
- *User Interface (UI) Proxy*: In order to show the results, the UI needs a proxy that consumes the data from the producer and sends it to the UI via a persistent web socket connection, using the socket.io⁶ framework. Also, this Node.js service uses express.js framework in order to expose a REST API, through which the UI can request data in a new search.
- *Web UI*: The web UI is built on top of the dc.js library, which uses the Big Data processing framework crossfilter.js and the data visualization library d3.js. This UI is composed of five parts: (a) The general DQ results; (b) The DQ results per dimension; (c) The visualization of the presence of a dimension in the stream, that is, the percentage of tweets with values for each dimension; (d) The Tweets vs. re-tweets part, comparing DQ values considering or leaving out re-tweets, respectively; and (e) The verified vs. unverified users part, indicating the DQ for tweets coming from verified or unverified users. Of course, this UI can be easily extended according to the analysis needs, to gain insight on the DQ of the data streams.

The data flow works as follows. First, the UI performs a request to the UI Proxy via the REST API in order to start a search using some query, indicating the *topic* ID to use (just a randomly-generated name), and some optional advanced parameters (a list of *completeness* properties to analyze, and the weights of each DQ metric). Then, the Proxy performs a REST API call to a Kafka producer service instance in order to start the ingestion of the feeds using those parameters. At this point, the Kafka producer service creates the topic

³ <https://kafka.apache.org/>.

⁴ <https://zookeeper.apache.org/>.

⁵ <https://projects.spring.io/spring-boot/>.

⁶ <https://socket.io/>.

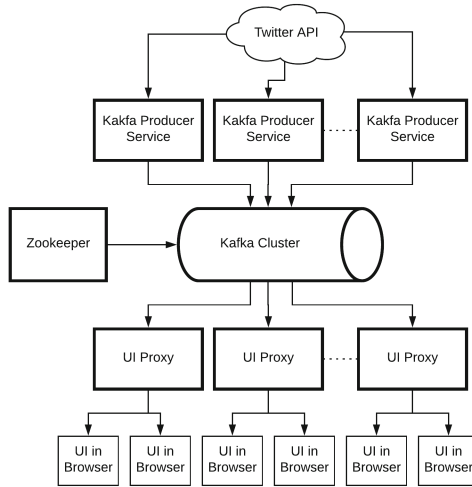


Fig. 1. Architecture diagram.

involved and starts a Kafka producer that fetches the feeds from the Twitter API and publishes its to the topic. After this initialization, the UI proxy starts a Kafka consumer peeking the feeds from the respective topic, and forwards the information to the UI. Finally, the UI processes the records using `crossfilter.js` and shows the data in real time using the `dc.js` library.

Remark 1. The system may scale horizontally as needed, just adding more Kafka producer services. To scale the Kafka Cluster more workers can be added, and Zookeeper will take care of their coordination.⁷

7 Experiments

This section describes the experiments performed over the implementation presented in Sect. 6, reports the results, and discusses them.

7.1 Use-Cases Description

The goal of the experiments is to illustrate how the quality of a Twitter stream can be measured using the dimensions and metrics presented above. Of course, there are countless ways in which the quality of data in tweets can be analyzed. The experiments presented here are just aimed at showing how the concepts discussed in previous sections can be studied using the tool presented in Sect. 6. The quality dimensions considered in all cases are: readability, completeness, and usefulness, with the metrics described in Sect. 4, and with the following weights: 0.5 for readability, and 0.25 for completeness and usefulness. Of course, the user can modify the weights according to her analysis needs. The dictionary used to

⁷ The system is available upon request to the authors.

check readability is given in [2] and contains 479,000 english words, including acronyms, abbreviations and even Internet slang. To compute sentiment (for usefulness), the Stanford CoreNLP software was used [9]. In all cases, the overall DQ of the stream is computed, as well as each DQ dimension individually, and the comparisons allowed by the UI (indicated in Sect. 6) are displayed. The experiments are aimed at:

- (a) Comparing the DQ of the whole stream of tweets, against the DQ of a stream filtered by a set of keywords related to some topic. The hypothesis is that the latter are more likely to have better quality than the former.
- (b) Performing the same comparisons above, but requesting the presence of different sets of properties (that is, changing the requested schema). This will give insight on which are the properties more likely to be present in a stream of tweets, and investigating the impact on this of the keyword filtering.
- (c) Determining if there is a correlation between the DQ of a stream, and the percentage of re-tweeted tweets that it contains. The hypothesis here is that a tweet with a high number of re-tweets is likely to be of high quality.
- (d) Comparing the quality of tweets from verified and not-verified users.

Next, the problems used to address the goals above, are described.

Problem 1. The first problem p_1 consists in analyzing the tweets in a stream, with no keyword filtering, i.e., all tweets provided by the Twitter API. The UI allows to indicate the set of properties considered for schema completeness. In this case, $props_{p_1} = \{\text{id, id_str, lang, retweet_count, usr, text, source}\}$.

Problem 2. For the second problem p_2 , the same set $props_{p_1}$ is used, but the stream is filtered using the keywords: $\{\text{Trump, Obama, Hillary}\}$.

Problem 3. The third problem p_3 consists in analyzing the tweets in a stream like in the previous problem, but considering a larger set of properties, namely all the ones supported by the UI. This allows to study how are the properties distributed in Twitter streams, that is, how many tweets contain the space coordinates or the language, for example.

Remark 2. Again, it must be clear that it is not intended here to draw conclusions on the DQ of Twitter feeds, but to suggest how the concepts and tools presented in this paper can be used to analyze such feeds.

7.2 Results and Discussion

Performance results were quite satisfactory. Tweets were captured and displayed at a rate of 2000 per minute (for non-filtered tweets), and at about 600 per minute, for filtered tweets, depending on how many tweets pass the filters.

The results obtained for the problems above are commented next, and illustrated by graphics. The figures show the status of some runs, such that after several thousands of consumed tweets, the results become stable, that is, the graphs do not change significantly.



Fig. 2. Results for problem 1.

Figure 2 shows a portion of the UI, displaying the results obtained from running the system with the conditions of *Problem 1*. In the upper part, the Y-axis represents the number of tweets, and the X-axis the DQ values, in intervals of 0.1. It can be seen that the general DQ is low (most of the tweets fall on the left part of the graph). Readability, completeness and usefulness are shown in the lower part. The first one is displayed as a boxplot, while the other two are displayed as bar charts. Since Readability is low, most of the values in the boxplot are outliers. About 55% of the tweets have values in all the fields in $props_{p_1}$, and more that 70% of the tweets have usefulness = 0. Results also showed (graphics omitted for the sake of space) that all DQ values are better when only re-tweeted tweets are considered, and for tweets posted by verified users.

Figure 3 displays the results obtained under the conditions of *Problem 2*, using the keywords Trump, Obama, and Hillary (an “OR” condition). The intention is to capture tweets with political content, based on the hypothesis that their quality should be better than for non-filtered tweets. It can be seen that the general DQ is much better, and on the upper part of the right-hand side, most of the tweets fall on the right part of the graph. Readability, completeness and usefulness are much better than in Fig. 2, and all DQ values (like in Problem 1) are better when only re-tweeted tweets are considered, and are also better for tweets posted by verified users.

Figure 4 displays, for each property supported by the UI, the percentage of tweets that contains values for that property. Only the values for Problem 1 are displayed, due to space limitations. Anyway, the set of properties that were checked by the user is the same for Problems 1 and 2. The darker portion of the circles indicates the percentage of tweets containing no value for the property.



These are the Data Quality results per dimension, taking into account both tweets and re-tweets.

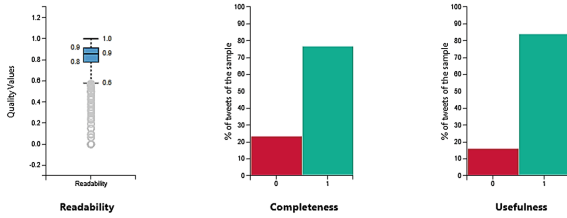


Fig. 3. Results for problem 2.



Fig. 4. Completeness for problem 1.

It can be seen that the `coordinates` field (at the top-right corner) has no value for any tweet, that is, no tweet is geo-referenced.

Figure 5, shows the results for Problem 3. Recall that in this problem, tweets are captured like in Problem 2, but the property set includes all properties supported by the system, for example, the spatial coordinates of the tweets.

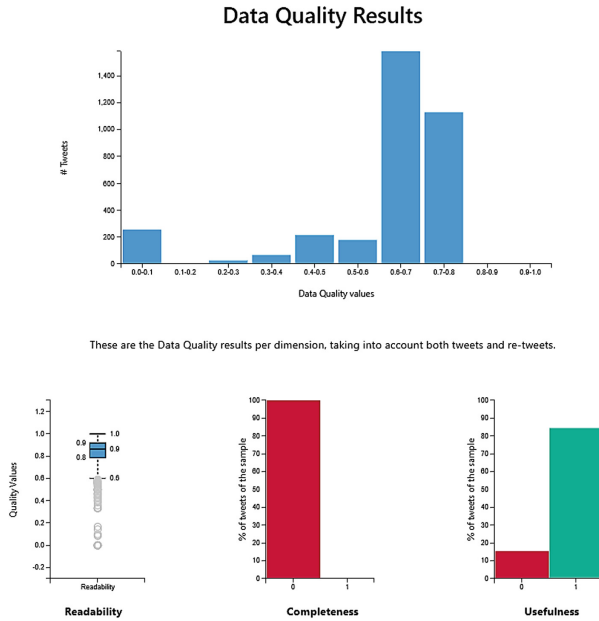


Fig. 5. Results for problem 3.

Figure 4 shows that this property is not satisfied by any tweet. Therefore, the completeness dimension for Problem 3 has value 0 (given that all properties are required to be present), which lowers the overall quality.

8 Conclusion and future work

This paper studied the particularities of assessing data quality in a Big Data context, and presented a system that allows analyzing such quality over Twitter streams in real time. Experiments performed over many different Twitter streams, showed how the concepts presented and tools developed could be applied in a real-world Big Data scenario. Still, there is plenty of room for further work. One line of research could be oriented to define more DQ dimensions and metrics for this or other settings (along the lines of [8]), since, as explained, this is typical context-dependent DQ. Also, new and more sophisticated visualization tools could extend and enhance the implemented framework. All of these will be part of future work.

Acknowledgments. Alejandro Vaisman was partially supported by the Argentinian Scientific Agency, PICT-2014 Project 0787.

References

1. Data, data everywhere (2008). <https://www.economist.com/node/15557443>
2. English-words project (2018). <https://github.com/dwyl/english-words>
3. Agrawal, D., Bernstein, P., Bertino, E., Davidson, S., Dayal, U.: Challenges and opportunities with big data (2011). <https://docs.lib.purdue.edu/cgi/viewcontent.cgi?referer=www.google.com.ar/&httpsredir=1&article=1000&context=cctech>
4. Batini, C., Rula, A., Scannapieco, M., Viscusi, G.: From data quality to big data quality. *J. Database Manag.* **26**(1), 60–82 (2015)
5. Batini, C., Scannapieco, M.: Data Quality: Concepts Methodologies and Techniques. DCSA. Springer, Heidelberg (2006). <https://doi.org/10.1007/3-540-33173-5>
6. Bolchini, C., Curino, C.A., Quintarelli, E., Schreiber, F.A., Tanca, L.: A data-oriented survey of context models. *SIGMOD Rec.* **36**(4), 19–26 (2007). <https://doi.org/10.1145/1361348.1361353>
7. Ciaccia, P., Torlone, R.: Modeling the propagation of user preferences. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (eds.) *ER 2011. LNCS*, vol. 6998, pp. 304–317. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24606-7_23
8. Firmani, D., Mecella, M., Scannapieco, M., Batini, C.: On the meaningfulness of “big data quality” (invited paper). *Data Sci. Eng.* 1–15 (2015)
9. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S.J., McClosky, D.: The stanford CoreNLP natural language processing toolkit. In: Association for Computational Linguistics (ACL) System Demonstrations, pp. 55–60 (2014). <http://www.aclweb.org/anthology/P/P14/P14-5010>
10. Marotta, A., Vaisman, A.: Rule-based multidimensional data quality assessment using contexts. In: Madria, S., Hara, T. (eds.) *DaWaK 2016. LNCS*, vol. 9829, pp. 299–313. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43946-4_20
11. Poeppelmann, D., Schultewolter, C.: Towards a data quality framework for decision support in a multidimensional context. *IJBIR* **3**(1), 17–29 (2012)
12. Scannapieco, M., Catarci, T.: Data quality under a computer science perspective. *Archivi Comput.* **2**, 1–15 (2002). <https://www.fing.edu.uy/inco/cursos/caldatos/articulos/ArchiviComputer2002.pdf>
13. Stefanidis, K., Pitoura, E., Vassiliadis, P.: Managing contextual preferences. *Inf. Syst.* **36**(8), 1158–1180 (2011)
14. Strong, D.M., Lee, Y.W., Wang, R.Y.: Data quality in context. *Commun. ACM* **40**(5), 103–110 (1997). <https://doi.org/10.1145/253769.253804>
15. Task Team on Big Data: Classification of types of big data (2007). <https://statswiki.unece.org/display/bigdata/Classification+of+Types+of+Big+Data>
16. Wang, R.Y., Strong, D.M.: Beyond accuracy: what data quality means to data consumers. *J. Manag. Inf. Syst.* **12**(4), 5–33 (1996)



Integrating Approximate String Matching with Phonetic String Similarity

Junior Ferri, Hegler Tissot, and Marcos Didonet Del Fabro^(✉)

C3SL Labs, Universidade Federal do Paraná, Curitiba, Brazil
junior.ferri@ufpr.br, hegler@gmail.com, marcos.ddf@inf.ufpr.br

Abstract. Well-defined dictionaries of tagged entities are used in many tasks to identify entities where the scope is limited and there is no need to use machine learning. One common solution is to encode the input dictionary into Trie trees to find matches on an input text. However, the size of the dictionary and the presence of spelling errors on the input tokens have a negative influence on such solutions. We present an approach that transforms the dictionary and each input token into a compact well-known phonetic representation. The resulting dictionary is encoded in a Trie that is about 72% smaller than a non-phonetic Trie. We perform inexact matching over this representation to filter a set of initial results. Lastly, we apply a second similarity measure to filter the best result to annotate a given entity. The experiments showed that it achieved good F1 results. The solution was developed as an entity recognition plug-in for GATE, a well-known information extraction framework.

Keywords: Entity recognition · Metaphone · Text tagging · Trie
Active nodes · Fast similarity search

1 Introduction

An Information Extraction (IE) pipeline is composed by tasks aiming at extracting information from unstructured sources and making it available in specific and structured formats [1, 12]. The Named Entity Recognition (NER) task aims at finding and classifying specific entities within a text, such as organizations, cities or drug names [4, 14].

Several approaches use well-defined dictionaries as input for NER tasks [16]. The dictionaries contain lists of classified entities. They are appropriate solutions either when the scope is limited and when there is no need to use machine learning techniques, or when they could be used as input for a machine learning solution. Existing solutions often use indexing structures such as suffix trees and arrays, q-grams or q-samples. However, it is not enough to use only index-like structures to support string exact matching algorithms. It is necessary to take into account the existence of spelling/typing errors in the input text, thus supporting Approximate String Matching [10]. The size of the dictionary may also be an issue, specially if there are several dictionaries used by the same task.

Approximate String Matching (ASM) has been widely studied in different contexts, including Information Extraction (IE) and in the Named Entity Resolution (NER) task. One of the central studies was the survey by Navarro et al. [10]. In this paper, however, we do not intend to be exhaustive in this context, but to narrow the scope with recent approaches that provide Trie-based solutions for Approximate String Matching.

Recent studies coupling Tries with inexact matching are the works from [5, 7]. They introduce the notion of valid (also called active) nodes while searching through the tree nodes. The active nodes have a calculated Edit Distance (ED) value which is lesser or equal than a max allowed ED value. They experimentally showed that these approaches were superior than q-Grams based solutions. In [8], the authors improve the active nodes computation, by incrementally computing them and storing them in a cache, creating the ICAN/ICPAN algorithms. The IncNGTrie [17] algorithm maintains a smaller set of active nodes, improving the performance. The META approach [3] presents a solution based on matching over compact indexes and that supports top-k queries. The most recent work from [15] handles approximate matching using efficient Trie implementations, though in the context of abbreviations, not full words. However, these approaches, or similar solutions, have not experimented using phonetic information from the text and dictionary, such as applying conversion using the Soundex [18] or Metaphone [11] algorithms. There are approaches that combine different similarity measures into a super-metric [6], though using distinct solutions than the Trie-based encoding, which has been shown to be effective.

In this paper we present an approach that couples a phonetic conversion algorithm with a Trie-based encoding in a NER task. First, we transform the given dictionary into a phonetic representation using the Metaphone algorithm using a well-known API. The phonetic representation is encoded in a Trie, and the Trie approximate matching is developed based on the active nodes algorithm from [5]. Each input token is also converted and then matched. The main advantage of this encoding is the reduced size, which may be important when using several dictionaries. It produces a Trie around 72% smaller than a Trie with the complete strings. In order to avoid a low precision, since the representation is smaller, we apply a second string similarity metric to return the best result, this time over the original string, linked by the Trie structure.

We have executed a set of experiments showing the applicability of this two-phase matching solution. The approach is implemented as a Gazeteer plug-in¹ for the GATE suite [2], a known information extraction framework. The implementation enables setting execution parameters, including the string metrics.

This paper is organized as follows. Section 2 presents our solution integrating inexact matching and a Trie-based ASM solution. Section 3 shows the experiments. Section 4 has the final conclusions.

¹ <http://gitlab.c3sl.ufpr.br/faes/asm/tree/master>.

2 Inexact Matching and Phonetic Encoding in a NER Task

In the following sections we describe the tree major steps on how we integrate inexact matching with phonetic encoding.

2.1 Input Tokens Extraction

We define the NER task as the following, adapting the definition from [14] to add the input dictionary. Given a document D , composed by a sequence of tokens $\mathbf{T} = \{t_1, \dots, t_n\}$, the NER task extracts from D a set of fields $\mathbf{F} = \{f_1, \dots, f_k\}$, where each field is an attribute-value pair $f_i = \langle a, v \rangle$. The value v is a token or a set of tokens matched with a key k , which has a corresponding label value lv , both available in entries from an input dictionary $ID = \{\langle k_1, lv_1 \rangle, \dots, \langle k_j, lv_j \rangle\}$. For instance, we could have a field $f_i = \langle City, London \rangle$, where the *City* label is extracted from a dictionary after matching an input token with the *London* key. In other words, it produces a list with annotated input tokens with labels from the input dictionary.

A token is a finite sequence of characters representing a subset of another finite sequence of characters, both conforming to the same alphabet. A token is extracted through the definition of delimiters d_{start} , d_{stop} to identify its start and end positions within a given sequence. Given a sequence of characters S , with size $|S|$, the size of each *token* t from T , is $0 < |t| \leq |S|$ and the sum of all tokens size is $\sum |T_{1..n}| \leq |S|$. The tokenizing rules depend on the tokenizer chosen. In our case study, we will use an existing tokenizer which is a plugin from the GATE framework.

Before the matching process, each individual token and the keys from the input dictionary are converted into a phonetic representation. We apply a phonetic conversion function:

$$CF(x : String) : String = t_{ph} \quad (1)$$

where t_{ph} is a new phonetic token. In a large part of dictionary-based NER solutions, there is no CF function, so they perform exact or approximate matching. In our solution, the matching process is done using only the newly produced phonetic tokens. The phonetic conversion function could be existing phonetic conversion algorithms, such as the Metaphone algorithm, which provides a compact phonetic representation for the English language. Table 1 presents some examples on how the Metaphone algorithm phonetically represents English words. We included samples in two common utilization of Gazeteers: cities and medication names.

2.2 Phonetic Approximate String Matching

The matching process to find a named entity considers only full tokens or set of full tokens. For example, if we have an input key *Brazil* and the current input

Table 1. Examples of phonetic representations resulting from Metaphone.

Word	Phonetic representation
Medroxalol	MTRKSSL
Amoxicillin	AMKSSLN
Bromfenac	BRMFNK
New York	NYRK
Avondale Estates	AFNTLSTTS
Washington	WXNKTN

token is *Brazilian*, the matching process does not consider this dictionary entry as a valid matching, if it is not within an Edit Distance (ED) limit.

We implement an algorithm that uses a Trie tree to encode the dictionary and to perform the inexact matching with the input tokens, using the phonetic converted versions of the tokens and the dictionary. Our algorithm implements the idea from [5], having a set of active nodes while searching the tree nodes, which have an Edit Distance lesser than a given threshold value. Briefly, the algorithm checks whether each character from an input token matches with a given Trie node and its descendents. If it does not match, the node ED is increased by 1. This search process stops when a node has the ED value larger than the given limit, when it is then deactivated. This need to be done for each previously activated node. The core algorithm is the same, with two main differences. First, the data structure has additional information to support NER matching, containing the following fields:

- **character**: the current character representing the node;
- **child**: an array containing the child nodes;
- **entryEnd**: defines if a given node identifies the end of a dictionary entry. It is used for composed entries;
- **activeNode**: identifies if the node is an active node, i.e., its assigned ED value is within a given threshold;
- **currentEd**: the current ED value;
- **entries**: stores the dictionary entries for a given node. One node may contain more than one dictionary entry. In this case the input token is annotated more than one time;
- **dictionaryEntry**: it contains the complete entry and the label used to annotate the input tokens.

As second difference, we add an extra step to allow the matching with entities composed with more than one token, for instance, a composite city name such as *New York*. This means that the matching is not performed only token by token, but after an unsuccessful token match, the algorithm continues to verify if it could match as a composed token. The result of the matching process is a list of candidate entities to be annotated. Utilizing compact phonetic structures diminishes the size of the Trie, however, it has the drawback of increasing the

number of matchings. For this reason, it is necessary to add a filtering step, which is explained in the following section.

2.3 Filtering the Results

The results from the matching phase are filtered by applying the following similarity function:

$$Sim(t : String, e : String) : Float = s \quad (2)$$

where t is the input token, e is the dictionary entry and s is a similarity value between both parameters, with $0 < s \leq 1$. However, to avoid any information loss, the parameters t and e are the original input token and entry, not the phonetic representation used along the previous step. The approach does not define a new similarity function, but it uses existing ones, such as the Jaro-Winkler or $Stringsim$ [13] metrics.

The result of the similarity function is used in two filtering rules. First, we set up a similarity threshold, called *minimum similarity*, to be considered (e.g., 0.7). Second, we rank the remaining entries and we choose only the best result.

The result of the filtering phase is a list with the annotated entities. Each entity contains the start and end positions in the input text, so it can be integrated with NER frameworks, the annotation labels associated and the final ED value.

3 Experiments

We have implemented a plug-in for the GATE suite to evaluate our approach, which is freely available for download,² as well as the dictionary, the input text and the complete raw results. In addition to the matching algorithm, we took special attention on providing a fully configurable plug-in, which means several configuration parameters can be easily modified. The main parameters are the following: *maxEditDistance*, *minSimilarityAccepted*, *similarityClass/Method*, *conversionClass/Method* (their names are self explanatory). We also provide parameters for setting up the tokenizer and the format of the dictionary, though we do not detail them here.

The input dictionary is a list with 76,912 English words obtained from the WordNet [9] database. The input entities with errors are randomly selected from a collection of common misspellings from Wikipedia.³ It contains the correct word and a version with a misspelling error. We used 1,000 input words, all with at least one misspelling error.

We applied our approach using two main settings. First, we used the Metaphone⁴ conversion function prior to the matching. Second, we used the original

² <https://gitlab.c3sl.ufpr.br/faes/asm/tree/master>.

³ https://en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings.

⁴ Implementation from the commons-codec-1.10.jar library, available at https://commons.apache.org/proper/commons-codec/download_codec.cgi.

tokens. We used the Jaro-Winkler metric for filtering the results.⁵ We have used the *Strings_{sim}* metric as second similarity metric, which keeps the similarity values higher for words with lower differences.

We used three different Edit Distance values: 0, 1 e 2. The 0 value was used to evaluate the effect of the conversion function on already eliminating errors. The limit of 2 was chosen because a higher value would produce a too low recall. For each ED value, we used 3 minimum similarities: 0.7, 0.8 and 0.9. Values smaller than 0.7 returned too many results. We evaluated the results in terms of: (a) *precision* - the fraction of the relevant annotations among all the returned ones; (b) *recall* - the fraction of relevant annotations among the total; and (c) the *F1 measure* - the harmonic average between both. We used the list with the correct words to verify these results.

3.1 Trie Construction

Table 2 shows a comparison between the size of the produced Trie using the original text (no phonetic conversion) and using the Metaphone function.

Table 2. Trie size for 76.912 entries

	Original text	Metaphone
Input chars	658.774	372.226
Avg. entries size	8.6	4.8
Trie nodes	240.484	67.602

The utilization of the Metaphone conversion on the list of 76.912 entries diminished the number of the input characters by 43%, from 658.774 to 372.226. After encoding the Trie, the number of nodes was 72% smaller, from 240.484 to 67.602. We do not measure the performance results on the Trie creation, since it is constructed only once. However, the reduced size yields loss of information. The impact of such codification is explained in the next section.

3.2 Phonetic Approximate Matching

The resulting scores are presented in Tables 3(a) to 4(b). First, we present the results when using the Metaphone conversion. Second, the results without a conversion function.

The choice of good parameters values is important to achieve good results. For instance, choosing a low ED value may minimize the choice of higher similarity thresholds, since it affects a lesser number of entries. Combined variations of ED and minimum similarity results yield results with a large variation. For instance, the recall values vary from 64.2% to 89.4%. The experiments results may be used to guide on the choice of the metrics and this threshold.

⁵ Implementation from lucene-suggest-5.2.1.jar, at <http://lucene.apache.org/>.

Table 3. Metaphone conversion

(a) Jaro-Winkler					(b) $String_{sim}$				
ED	Min Sim	Precision	Recall	F1	ED	Min Sim	Precision	Recall	F1
0	0.7	81.3%	64.2%	71.7%	0	0.7	86.7%	66.5%	75.3%
0	0.8	84.4%	64.2%	72.9%	0	0.8	90.7%	66.3%	76.6%
0	0.9	87.8%	62.7%	73.2%	0	0.9	94.9%	51.9%	67.1%
1	0.7	81.5%	84.4%	82.9%	1	0.7	85.7%	89.5%	87.6%
1	0.8	81.5%	84.4%	82.9%	1	0.8	86.4%	89.3%	87.8%
1	0.9	82.6%	82.8%	82.7%	1	0.9	89.3%	72.9%	80.3%
2	0.7	78.3%	82.3%	80.3%	2	0.7	84.0%	88.7%	86.3%
2	0.8	78.7%	82.3%	80.4%	2	0.8	84.0%	88.5%	86.2%
2	0.9	79.7%	81.5%	80.6%	2	0.9	87.3%	73.0%	79.5%

We can see that the highest precision (87.8%) from Table 3(a) was obtained with the ED value equals to 0. This means that the choice of this phonetic representation also had an impact and it absorbed partially the misspelling errors, since every word had at least 1 error. However, the recall was smaller, 62.7%, obtaining an F1 score of 73.2%. Increasing the ED by 1 had a positive impact on the recall, since we had 84.4% of recall. This combination yielded the best F1 score, 82.9%. The precision diminished on about 2% when augmenting the ED. In addition, the execution time of the search in the tree started on around 15 ms, raising gradually for higher EDs.

Without the phonetic conversion, as shown in Table 4(a), the best precision was obtained with $ED \leq 1$ (86.6%), with the same result with 3 filtering similarity metrics. The best precision was obtained with $ED \leq 2$. The best recall was obtained with a similar configuration than with the conversion. Finally, the best F1 results were obtained with $ED \leq 2$ and with filtering similarity ≥ 0.7 and ≥ 0.8 . A higher filtering value yielded a too low precision. We do not consider ED equals to 0, since the result is an exact matching. The execution times have a similar order of magnitude, independently from using the Metaphone or without conversion. This is an interesting result since we could argue that the phonetic conversion could be chosen without additional time cost.

The results from $String_{sim}$ were obtained with a very similar choice of parameters, for both cases. However, the filtering metric yielded a better precision in both cases, 87.8% with conversion and 88.2% without conversion. The performance results were slightly worse, though not very significant.

We have also conducted experiments using the Double Metaphone conversion. This method has an even more compact representation: the Trie size is about 95% smaller, with 10,325 nodes, since in this representation the tokens have only 4 characters. However, it increases too much the number of the matched entries, thus the filtering metrics need to be executed over too many entries. For this reason, we discarded such approach.

Table 4. No phonetic conversion

(a) Jaro-Winkler					(b) <i>Strings_{sim}</i>						
ED	Min	Sim	Precision	Recall	F1	ED	Min	Sim	Precision	Recall	F1
1	0.7	86.6%	72.0%	78.6%	78.6%	1	0.7	89.3%	74.3%	81.1%	81.1%
1	0.8	86.6%	72.0%	78.6%	78.6%	1	0.8	89.5%	74.3%	81.2%	81.2%
1	0.9	86.6%	70.7%	77.9%	77.9%	1	0.9	91.8%	65.3%	76.3%	76.3%
2	0.7	82.6%	84.4%	83.5%	83.5%	2	0.7	87.0%	89.4%	88.2%	88.2%
2	0.8	82.6%	84.4%	83.5%	83.5%	2	0.8	86.9%	89.2%	88.1%	88.1%
2	0.9	82.6%	82.8%	82.7%	82.7%	2	0.9	88.8%	73.1%	80.2%	80.2%

To summarize, we can see from these tables that the parameters with better F1 were obtained with ED equals to 1 and 2, and with a small variation in the filtering similarity threshold. Comparing the three methods, the best F1 scores were better with the *Strings_{sim}* metric. This indicates that a phonetic approach with filtering is a valid solution when choosing the right parameters and filtering function. The performance results were similar from both metrics and conversion methods. Considering the phonetic conversion, the increase on precision was not very high and the performance results were similar. However, the smaller size of the Trie justifies the choice of a conversion method. These findings shows that the initial matching on the phonetic version acts like an initial filter, in order to reduce the size of the entries.

4 Conclusions

We presented an hybrid approach that integrates approximate string matching with phonetic string similarity. We have implemented a version of an existing algorithm to encode a Trie and to apply inexact string matching. We changed the input of the algorithm by using a compact phonetic representation, using the Metaphone algorithm. The size of the Trie was 72% smaller, thus having a large impact on the Trie size. The Trie inexact phonetic matching acts like an initial filter of similar words, and a second similarity metric, applied on the original text, does the final filtering. The best F1 scores were achieved with the *Strings_{sim}* metric. This means it can be used when the size of the Trie is important (for instance, when there are several dictionaries used on a desktop application). We have seen that the choice of the right parameters is also important, otherwise the F1 scores may decrease. The phonetic conversion did not have a significant impact on performance, making it a good choice.

We have implemented the approach as a Gazeteer plug-in for GATE, a well-known information retrieval framework, with setup parameters that can be modified, including the metrics presented. This flexibility can be used to conduct further experiments with different settings, from the initial tokenizer, the execution parameters, up to the phonetic algorithm and the similarity measure, in

order to obtain better final F1 scores in a future work. All the results and the plug-in are freely available.

Acknowledgments. This work was partially funded by Project *Sistema de Monitoramento de Políticas de Promoção da Igualdade Racial (SNPPIR)*.

References

1. Cunningham, H.: Information extraction, automatic. In: Encyclopedia of Language and Linguistics, 2nd edn. (2005)
2. Cunningham, H., Tablan, V., Roberts, A., Bontcheva, K.: Getting more out of biomedical documents with gate's full lifecycle open source text analytics. PLOS Comput. Biol. **9**(2), e1002854 (2013)
3. Deng, D., Li, G., Wen, H., Jagadish, H.V., Feng, J.: Meta: an efficient matching-based method for error-tolerant autocompletion. Proc. VLDB Endow. **9**(10), 828–839 (2016)
4. Grishman, R., Sundheim, B.: Message understanding conference-6: a brief history. In: COLING, vol. 96, pp. 466–471 (1996)
5. Ji, S., Li, G., Li, C., Feng, J.: Efficient interactive fuzzy keyword search. In: Proceedings of the 18th WWW, WWW 2009, Madrid, Spain, pp. 371–380. ACM (2009)
6. Lamontagne, L., Abi-Zeid, I.: Combining multiple similarity metrics using a multicriteria approach. In: Roth-Berghofer, T.R., Göker, M.H., Güvenir, H.A. (eds.) ECCBR 2006. LNCS (LNAI), vol. 4106, pp. 415–428. Springer, Heidelberg (2006). https://doi.org/10.1007/11805816_31
7. Li, G., Ji, S., Li, C., Feng, J.: Efficient type-ahead search on relational data: a tastier approach. In: Proceedings of the 2009 ACM SIGMOD, SIGMOD 2009, Providence, Rhode Island, USA, pp. 695–706. ACM (2009)
8. Li, G., Ji, S., Li, C., Feng, J.: Efficient fuzzy full-text type-ahead search. VLDB J. **20**(4), 617–640 (2011)
9. Miller, G.A.: WordNet: a lexical database for English. Commun. ACM **38**(11), 39–41 (1995)
10. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. **33**(1), 31–88 (2001)
11. Philips, L.: Hanging on the metaphone. Comput. Lang. Mag. **7**(12), 38–44 (1990)
12. Sarawagi, S.: Information extraction. Found. Trends Databases **1**(3), 261–377 (2008)
13. Tissot, H., Peschl, G., Del Fabro, M.D.: Fast phonetic similarity search over large repositories. In: Decker, H., Lhotská, L., Link, S., Spies, M., Wagner, R.R. (eds.) DEXA 2014. LNCS, vol. 8645, pp. 74–81. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10085-2_6
14. Culotta, A., Kristjansson, T., McCallum, A., Viola, P.: Corrective feedback and persistent learning for information extraction. Artif. Intell. **170**(14), 1101–1122 (2006)
15. Stonebraker, M., Tao, W., Deng, D.: Approximate string joins with abbreviations. Proc. VLDB Endow. **11**(1), 53–65 (2017)
16. Wimalasuriya, D.C., Dou, D.: Ontology-based information extraction: an introduction and a survey of current approaches. J. Inf. Sci. **36**(3), 306–323 (2010)
17. Xiao, C., Qin, J., Wang, W., Ishikawa, Y., Tsuda, K., Sadakane, K.: Efficient error-tolerant query autocompletion. VLDB Endow. **6**(6), 373–384 (2013)
18. Zobel, J., Dart, P.: Phonetic string matching: lessons from information retrieval. In: The 19th SIGIR, SIGIR 1996, Zurich, Switzerland, pp. 166–172. ACM (1996)

**Distributed Data Platforms, Including
Cloud Data Systems, Key-Value Stores,
and Big Data Systems**



Cost-Based Sharing and Recycling of (Intermediate) Results in Dataflow Programs

Stefan Hagedorn^(✉) and Kai-Uwe Sattler

Databases and Information Systems Group, TU Ilmenau, Ilmenau, Germany
{stefan.hagedorn,kus}@tu-ilmenau.de

Abstract. In data analytics, researchers often work on the same datasets investigating different aspects and moreover develop their programs in an incremental manner. This opens opportunities to share and recycle results from previously executed jobs if they contain identical operations, e.g., restructuring, filtering and other kinds of data preparation.

In this paper, we present an approach to accelerate processing of such dataflow programs by materializing and recycling (intermediate) results in Apache Spark. We have implemented this idea in our Pig Latin compiler for Spark called *Piglet* which transparently supports both, merging of multiple jobs as well as rewriting jobs to reuse intermediate results. We discuss the opportunities for recycling, present a profiling-based cost model as well as a decision model to identify potentially beneficial materialization points. Finally, we report results of our experimental evaluation showing the validity of the cost model and the benefit of recycling.

1 Introduction

Today, scalable distributed analytics platforms like Hadoop, Spark or Flink are typically used together with higher-level languages, either a domain-specific language (DSL) often integrated with a programming language (e.g. Scala or Python) or dedicated high-level dataflow languages such as Pig Latin or Jaql which can be extended by UDFs. Particularly, the latter ones offer the advantage of allowing to write complex analytical pipelines in an easy and incremental manner. Furthermore, this interactive and incremental formulation is also supported by notebook-like interfaces such as Jupyter or Zeppelin.

For discussing the opportunities for sharing and recycling intermediate results of dataflow programs in Apache Spark we consider the following exemplified use case: A (small) group of data scientists has to analyze several datasets containing sensor data from weather and environmental observation stations. These datasets contain spatio-temporal data but also metadata describing e.g. sensors. By analyzing this data, models are constructed to represent phenomena which can be

S. Hagedorn—This work was partially funded by the German Research Foundation (DFG) under grant no. SA782/22.

used for forecasting, classification etc. Main tasks in this process are integrating and preparing data (e.g. cleaning, transformation, and reduction), selecting relevant subsets (e.g. by time or region of interest), and applying machine learning and other analytical operations. Typically, this follows a rather incremental and explorative approach where the dataflow jobs are specified and executed step by step to inspect and validate results, test different parameters, and decide about subsequent steps to add further necessary operators. Furthermore, multiple scientists might use the same datasets in parallel or simply run multiple jobs to analyze different aspects. Based on these assumptions, there exist two obvious opportunities for sharing work:

merge: Merge a batch of submitted dataflow programs into a single job so that common parts are executed only once.

materialization: Explicitly (by user) or implicitly (i.e. automatically) insert save/load actions to recycle intermediate results across jobs, similar to materialized views.

Recycling results is especially useful when computation power is expensive or limited, but storage is cheap and available to a large extent. This is often the case for rented cluster resources on cloud providers like Amazon, Google, or Microsoft Azure. For example, an Amazon Elastic MapReduce cluster has a per-hour pricing of around \$ 2 (c4.8xlarge, Region Frankfurt) but storage is currently only \$ 0.024 per GB (S3, Region Frankfurt). If the execution times of the jobs running on such clusters can be reduced significantly, users could save a considerable amount of money. Inspired by the concept of materialized views and adaptive indexing, we present in this paper an approach on materializing and recycling (intermediate) results in Apache Spark-based dataflow programs. The goal of our work is a transparent materialization and reuse of intermediate results to unburden the data scientist from decisions about costs and benefits of materializing results, aggregations, and potentially also indexes. For this purpose, we propose a cost-based decision model which relies on profiling information obtained by instrumenting the Spark¹ runtime environment.

The contribution of our work is twofold: (1) We present and evaluate strategies for transparently materializing and reusing intermediate results of dataflow programs for platforms like Spark. (2) We present a cost and decision model for materialization points leveraging platform features and code injection for runtime profiling.

2 Related Work

Caching and reusing intermediate query results has been extensively studied for relational databases and data warehouses. Selecting partial results, or views respectively, for materialization [4, 7] as well as rewriting queries using these

¹ <https://spark.apache.org/>.

views [6] are two closely related problems that are used in database systems to improve query response time.

Early works on reusing materialized views (or derived relations) were done by Larson and Yang in [9, 18]. Other research results on the view-matching problems for SQL queries were published in [1] or [15]. In [11] the Hawc architecture is introduced that extends the logical optimizer of an SQL system and considers the query history in order to decide which intermediate result may be worth materializing to speed up further executions – even if this would create a more expensive plan which, however, is executed only once. A related problem is automatic index tuning which has been studied extensively [8, 12, 13, 16]. Here, recommenders analyze the given workload and underlying data and recommend to or autonomously create and drop indexes.

For Hadoop MapReduce the MRShare framework [10] merges a batch of jobs into a new batch of jobs so that groups of jobs can share scans over input files and the Map output. This is similar to our merging strategy described earlier. Other projects such as ReStore [5], PigReuse [2], or [17] are similar to MRShare in the sense that they all merge a batch of scripts into a single plan or share the intermediate results after a map phase. In PigReuse, the optimization goal is to minimize the number of operators and the number of generated MapReduce jobs - but they do not analyze the total cost of the generated plans.

For Spark several additional frameworks were created to support data analysts with their tasks. KeystoneML [14] is able to identify expensive operations in machine learning pipelines on Big Data platforms like Apache Spark. They employ a cost model using cluster costs (such as network bandwidth, CPU speed, etc.) and operator costs to estimate total execution costs. From this physical operators for a logical plan are chosen and materialization points are determined. RDDShare [3] is also based on Spark and simply identifies common operators in a batch of Spark programs and merges them into a single program.

Our work differs from the mentioned approaches in a way that they either only focus on merging a batch of submitted scripts into a single job or they do not use a cost model for their algorithms. Furthermore, most related work is based on Hadoop MapReduce, except KeystoneML and RDDShare, which differs significantly from Spark’s characteristics. However, KeystoneML focuses on choosing the best physical implementation of a logical operator and RDDShare only tries to merge a batch of scripts without reusing intermediate results over different runs.

3 Architecture Overview

Figure 1 shows the general architecture overview of our approach. Independently from the used language (Pig Latin, Scala, Python) and platform (Spark, Flink, Hadoop MapReduce), a dataflow program can be represented as a directed acyclic graph (DAG), where operators are the nodes and the directed edges are the links between these nodes that represent the dataflow. An optimizer component receives the DAG for the current job and after applying general

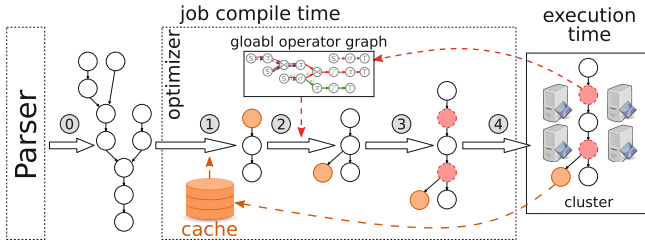


Fig. 1. Architecture overview: (0) Transform script to DAG, (1) Insert **LOAD** for existing data, (2) Insert **STORE** (based on statistics), (3) code instrumentation for profiling, (4) execute as Spark job. (Color figure online)

rule-based optimizations, the DAG will be modified for recycling. We employ a cache that will store the materialized results. Ideally, the cache should have access to HDFS for persistent storage. If materialized data exists for a part of the current DAG, we first will replace that part with a **LOAD** operator that reads the cached data. In the next step, the global operator graph (see Sect. 4.3) that stores profiling information is checked to determine if operators of the current DAG should be materialized and respective **STORE** operators are inserted. After that, we insert profiling operators to collect runtime statistics of the operators in the graph. During execution on a cluster, those operators will send information to the optimizer which will update its statistics. If intermediate results are to be materialized, those will be written to the cache by the inserted **STORE** operators.

We implemented the described cost-based decision model and profiler into our Piglet project: a parser and code generator from Pig Latin to Spark and other platforms. The code including the implementation of the cost model is available at our GitHub repository². However, we would like to emphasize that the model described in this paper is neither restricted to Piglet nor Spark and could easily be adopted into other platforms. The details of the decision model that uses the information in the global operator graph as well as the cache will be explained in more detail in the next sections.

4 A Cost-Based Decision Model

For our work we assume that in the DAG each operator o has an unique lineage identifier $lid(o)$. This lid consists of the operator name (**LOAD**, **FILTER**, **GROUPBY**, ...) and parameter values (e.g. filter predicates) together with the lid of its direct predecessor(s). This lineage identifiers provide a simple way to decide if two operators are identical, i.e. read the same input and produce the same output. Currently, we use this approach as a simple replacement for query containment check which is hard or even undecidable in case of user-defined code. The goal of our cost model is to identify those operators in the DAG where materializing their intermediate results speeds up subsequent executions most. Figure 2 shows

² <https://github.com/dbis-ilm/piglet>.

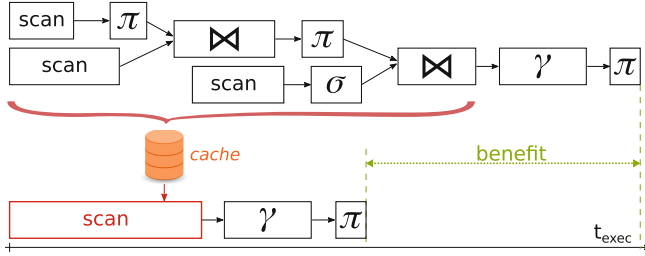


Fig. 2. Runtime difference for loading materialized result. (Color figure online)

a DAG where the width of a node’s box represents its processing time. If, e.g., the result of the second join operator is materialized, subsequent executions of dataflow programs that also contain this part in their respective DAG will benefit by only having to load the already present result from disk. This leads to two basic questions that need to be answered:

- (1) If multiple materialized results are applicable for reuse in a job, which of them should be loaded?
- (2) The intermediate result of which operators in the current job are worth materializing so that a subsequent execution will benefit most?

In order to support these decisions, our model introduces materialization points, for which the benefits are calculated.

4.1 Materialization Point

A materialization point M is a logical marker in a DAG denoting a position for the decision model to write or load the materialized results. Here, we distinguish between *candidate materialization points* and *materialization points*. Candidate materialization points are those potential places in a DAG, where the intermediate result should either be materialized or could be loaded from storage. We denote the (candidate) materialization point representing the output of operator o_i by M_i , meaning that M_i is an alias for the output of operator o_i in the optimizer component. Every non-sink operator can be regarded as a candidate materialization point. Obviously, one cannot achieve any benefit from materializing the result of a source operator and thus, these materialization points need not be considered. We denote the set of all materialization points M_1, M_2, \dots, M_n which are currently kept in the cache as the *materialization configuration* $\mathcal{M} = \{M_1, M_2, \dots, M_n\}$.

The decision model has to determine if the result of the respective operator is worth materializing or already materialized results can be loaded. Thus, materialization points are always a subset of these candidates.

4.2 Benefit

The decisions are based on the *benefit* regarding the execution time of the complete job with the main goal to minimize the overall execution time. Hence, the benefit is the amount of time saved when intermediate results can be loaded instead of executing the complete job, as depicted in Fig. 2. Alternatively, one can also regard the benefit as the amount of money saved by needing to rent fewer machines in a public cloud.

To calculate the benefit, the decision model is based on the actual costs of operators which are measured during execution of a job. For an operator o_i uniquely identified by its $\text{lid}(o_i)$ the following statistics are collected:

- **cardinality** $\text{card}(o_i)$: Number of result tuples of o_i .
- **tuple width** $\text{width}(o_i)$: The average number of bytes per result tuple of o_i .
- **execution time** $t_{\text{exec}}(o_i)$: Duration it takes the operator to completely process its input data.

The benefit of a materialization point M_i can be expressed as in Eq. 1.

$$t_{\text{benefit}}(M_i) = t_{\text{total}}(M_i) - t_{\text{read}}(M_i) \quad (1)$$

$$t_{\text{total}}(M_i) = \sum_{o \in \text{prefix}(M_i)} t_{\text{exec}}(o) \quad (2)$$

$t_{\text{total}}(M_i)$ denotes the cumulative execution time of operators in the prefix of o_i from the source to M_i and $t_{\text{read}}(M_i)$ is the time required to read the materialized data of M_i . If the prefix of M_i does not contain a join (or cross, etc.) $t_{\text{total}}(M_i)$ can be calculated as in Eq. 2. If the prefix of M_i does contain a join (or similar) operator j , with $k_1(j), \dots, k_n(j)$ as the direct inputs to j , only the longest (concerning execution time) of those branches is considered:

$$t_{\text{total}}(M_i) = \max\{t_{\text{total}}(k_1(j)), \dots, t_{\text{total}}(k_n(j))\} + \sum_{\substack{o \in \text{prefix}(M_i) \\ \wedge o \notin \text{prefix}(j)}} t_{\text{exec}}(o) \quad (3)$$

This means to take the maximum time of all input branches of the join operator and add the execution times of all other operators in the prefix of M_i which are not part of the join input, i.e. are located between j and o_i . The time to read the materialized result of M_i is calculated as:

$$t_{\text{read}}(M_i) = \frac{\text{card}(o_i) \cdot \text{width}(o_i)}{\text{bps}} \quad (4)$$

The factor bps stands for the number of bytes that can be read per second and depends on the cluster setup as well as the underlying hardware and thus, is an installation-specific calibration factor.

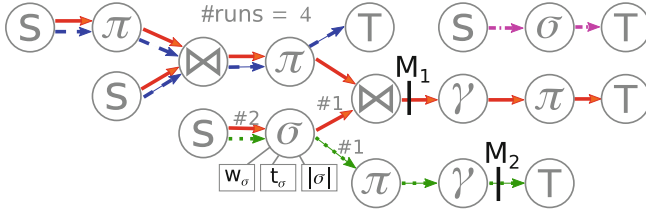


Fig. 3. The global operator graph for four jobs. Stored information on nodes and edges is exemplarily shown for a single operator node. (Color figure online)

4.3 Global Operator Graph

The benefit of each candidate materialization point is estimated based on execution of multiple jobs containing the corresponding operator instances. For this purpose, the statistics are maintained in a *global operators graph*, a DAG that was created by merging the DAGs of all ever submitted jobs. The graph is persistently stored and, therefore, available across multiple jobs.

With each operator instance, identified by its lid, the collected runtime statistics are associated while with each edge its frequency of occurrence in all executed jobs is stored. Additionally, the global operator graph also contains the materialization points, i.e., statistics about the already executed operators and materialized results. Figure 3 shows an example of such a global operator graph for four jobs J_1 (solid red edges), J_2 (dashed blue), J_3 (dotted green), and J_4 (dashed purple)³.

After the execution of a job finished and all statistics (runtimes and result sizes of the operators) have been collected (cf. Sect. 5), they are added to the respective nodes in the graph. If the operator is executed for the first time, no statistics are present for this operator and the collected values are simply added to the node in the graph. On the other hand, if the operator was already executed before as part of another job, present statistics are merged with the newly collected ones by averaging them.

The graph serves as input for the decision model and is used to calculate the benefit based on the statistics and materialization points.

4.4 Decision Model

The decision model is used to answer the two question posed in the beginning of this section.

Loading Existing Materialized Data. Answering the first question is straightforward. From the list of candidate materialization points for a given job, only those are selected for which materialized results are present. Then,

³ In reality, in the model there is only one edge between the nodes. The multiple edges are just for illustrating the different jobs.

from these candidates, the one materialization point that will result in the highest benefit is chosen to achieve the greatest speedup. If the job contains multiple paths, which are then combined in a join (or similar), selection of multiple materialization point, one for each path, is possible.

Materializing Intermediate Results. The decision model has three dimensions to consider when choosing a materialization point to actually write to persistent storage.

- (1) Which candidate materialization points should be selected for further investigation?
- (2) From the list of candidate materialization points resulting from (1), which of those should be materialized?
- (3) If the persistent storage is limited in space, decide which existing materialized result has to be deleted. Note, that due to space limitations, we do not consider the dimension of cache eviction strategies here and assume an infinite cache.

Selection of Candidate Materialization Points. Obviously a sink operator is not a candidate for materialization as the result is either written to persistent storage anyway or printed to screen. In the latter case, the materialization point before the that sink operator would be one whose result could be re-used by another job. A source operator will only read data from storage and pass it to the next operator without modification. Hence, materializing the output of a source operator will write the same data back to disk and no benefit can be gained from this. Therefore, subsequent operations only need to consider candidate materialization points that do not belong to a source or sink operator.

Ranking Materialization Points. To decide which materialization point to really materialize, different strategies exist, which might be suitable for different use cases:

- **latest:** A naïve but intuitive strategy for selecting a materialization point is to always choose the last possible one. This is the one materialization point that is closest to a sink operator. If a job contains n sinks, the materialization point before each sink is selected, which means to write n intermediate results. This is a simple caching strategy and might work well during the incremental development of scripts, described earlier. However, this bears the “risk” that the materialized result will not be needed again, e.g., if subsequently executed scripts are not the next step of the incremental development, but branch off at another operator so that an earlier materialization point would have been a better choice. Furthermore, the last materialization point may only bring a small (or even no) benefit for reusing.
- **maxbenefit:** Therefore, another option is to choose that one materialization point with the highest benefit. Compared to the previous strategy, where the last one is selected, it is guaranteed to bring the best possible benefit when

the result is needed again. Like in the previous strategy, if the result is not needed again, materialization was pointless.

- **markov:** Thus, selection of the materialization points should consider the probability for reuse – for which a Markov chain can be applied. In fact, we should regard this as a two dimensional (probabilities and benefits) optimization problem to maximize the benefit as well as the probability for reuse of the selected materialization points. This optimization problem is known as the Skyline or Pareto efficiency. The result of this optimization problem are all points where there exists no other point with both a higher benefit and higher probability. All points in the Pareto front mark materialization points with either a high probability and/or high benefit, thus being worth materializing. If only one materialization point should be selected, it has to be chosen from the Pareto front. For this, the probability of re-occurrence of a materialization point can be considered as the weight for the benefit, so that the materialization point with the highest product of probability and benefit should be selected:

$$\{M_i \in \mathcal{M} \mid \nexists M_j \in \mathcal{M}, i \neq j : P_{total}(M_j) * t_{benefit}(M_j) > P_{total}(M_i) * t_{benefit}(M_i)\} \quad (5)$$

If this set contains multiple elements, one can be chosen arbitrarily or user specified weights can be applied to express a favor of one dimension over the other. $P_{total}(o_i)$ denotes the minimum probability found on the path in the DAG from the source operators to o_i :

$$P_{total}(o_i) = \min\{P_{o_k, o_l} \mid o_k, o_l \in prefix(o_i), o_k \rightarrow o_l\} \quad (6)$$

where $o_k \rightarrow o_l$ means that o_l is a direct successor of o_k in the DAG. P_{o_k, o_l} describes the probability that o_k will be followed by operator o_l and can be calculated in different ways. One approach is to put the frequency into relation of the total number of executions, with respect to some time window \mathcal{W} . The probability P_{o_k, o_l} then would be as in Eq. 7

$$P_{o_k, o_l} = \frac{f_{o_k, o_l}^{\mathcal{W}}}{\min(\mathcal{W}, runs)} \quad (7)$$

$$P_{o_k, o_l} = \frac{f_{o_k, o_l}^{\mathcal{W}}}{deg_{\mathcal{W}}^+(o_k)} \quad (8)$$

A second approach is to put the frequency in relation to the possible other frequency operators that follow o_k , as shown in Eq. 8. Here, $f_{o_k, o_l}^{\mathcal{W}}$ is the plain frequency count for the transition stored on the edges, that lie within the considered window \mathcal{W} , $runs$ is the total number of jobs that are executed by the system, and $deg_{\mathcal{W}}^+(o_k)$ is the outdegree of a node o_k .

5 Profiling Dataflow Programs

Result Size. Unlike traditional DBMS, in the Big Data field one often works with plain text files (log files, csv, ...) and no central data management system that

has access to detailed statistics. Thus, in order to gain the desired information, there are two options: (1) The optimizer is started separately to analyze the input file and create a profile. Before executing a job, the optimizer then tries to come to a decision based on the statistics and selectivity estimations of the involved operators. (2) The job is instrumented with code that collects the necessary statistics during execution and the runtime or execution platform has to be extended by such an optimizer that manages and utilizes the collected data. The first approach is more or less what traditional DBMS do and is currently also being implemented in Apache Spark for Spark SQL. However, the second approach has the advantage that execution time is measured as well as the result size, instead of relying on estimations that are based on assumptions. In our evaluation we will show that the instrumentation does not incur in any significant overhead. To determine the total number of bytes in the result of an operator, and thus the size of the materialization point, we use Spark’s `SizeEstimator` that estimates the number of bytes for a given object. We sample the result of the operator and pass each result tuple individually into the estimator. The information for each partition is accumulated using Spark’s accumulator mechanism and send to the optimizer. From the received information the optimizer can calculate the average tuple size as well as the total number of tuples in the result.

Execution Time per Operator. A more difficult task is to measure the execution time of an operator. In Spark, a job is divided into stages, where each stage contains a sequence of operators that can be executed without data shuffling. Shuffling happens when an operation needs data from several partitions, e.g., `COGROUP`. Operators in the same stage can be executed in one scan over the partition. Spark comes with a `SparkListener` interface that provides information about status of the current execution including start and completion time of the stages that form the job. However, relying on the execution times of the stages is too coarse for our goal as possible materialization points for recycling would be after a stage only. Thus, there would not be many of such materialization points and more importantly we would lose most operators that are shared between different jobs, because they are hidden inside a stage and thereby reducing the usefulness of the idea. We therefore implemented our own approach, based on code instrumentation, to measure the execution duration of an operator per partition. On the logical level, *timing* operators are inserted between all other operators in the plan, as depicted in Fig. 4.

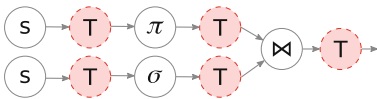


Fig. 4. Timing operators inserted into dataflow plan. (Color figure online)

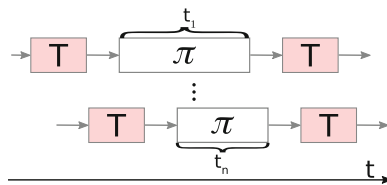


Fig. 5. Parallel execution of operations. (Color figure online)

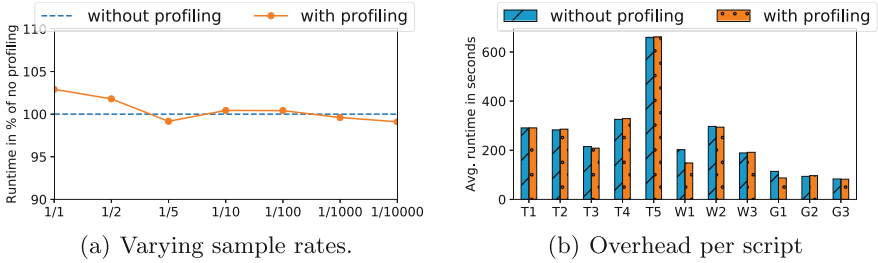


Fig. 6. Overhead of code instrumentation. (Color figure online)

The task of the timing operators is to send a message to the profiling manager component of the optimizer with the current system time, when they are executed⁴. The profiling manager will receive a timestamp and the lid of the according operator for each partition and calculates the average execution time of the operators based on this information.

The realization of this concept needs to deal with Spark’s lazy evaluation as well as the data parallelism. Thus, for each timing operator we inject code to report the current time when a partition is processed (using `mapPartitions`). The partitions that each operator instance processes can be of different sizes, although the platforms try to keep them balanced to avoid skewed workload on the nodes. Thus, the operator instances require different times to process their input. For n partitions, it results in n different execution time information, from which we have to derive an overall execution time for an operator (cf. Fig. 5). In our approach we use the average of these n collected times. Other strategies (min, max, median), however, are also possible and *min* or *max* could be used to implement an optimistic or pessimistic behavior.

Since an RDD has information about its parents, it is enough to only insert this code after the operator and let the profiling manager calculate the execution duration, based on the received times of the respective parent operator.

6 Evaluation

To evaluate the proposed decision model we make the following three hypotheses: (1) Injected profiling code only introduces a negligible overhead. (2) The materialization decision always improves query execution time and avoids bad decisions. (3) The best strategy for selecting a materialization point for writing depends on the workload setting.

We use real world data from three use case scenarios: **weather** contains sensor data from the SRBench [19] benchmark for hurricane Katrina (180 mio. tuples, 34.7 GiB), New York **taxi** trip data⁵ (Yellow Cabs, 2013–2016, 550 mio. tuples,

⁴ This requires that the clocks on all nodes are synchronized, of course. For example via NTP.

⁵ http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.

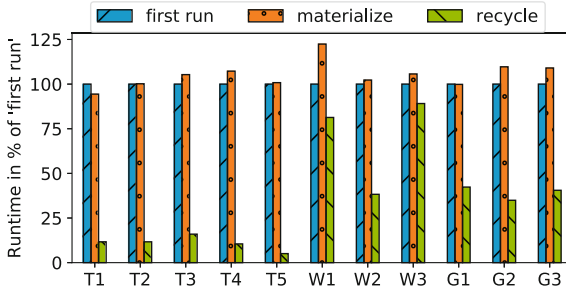


Fig. 7. Three executions times for each script showing the benefit of loading materialized results. Runtime in percent of the first execution. (Color figure online)

90 GiB) together with New York block data⁶ (38,000 tuples, 18 MiB), and the GDEL⁷ data from 2013 to 2016 (127 mio. tuples, 48 GiB). We created several scripts for each use case scenario: T1–T5 for taxi scenario, W1–W3 for weather scenario, and G1–G3 for GDEL. The scripts and their DAG visualizations can be found in our GitHub repository (see footnote 2). Our Spark cluster consists of 16 nodes with: Intel Core i5 2.90 GHz, 16 GB DDR3 RAM, 1 TB disk, 1 GBit/s LAN. The cluster runs Hadoop 2.7, Spark 2.0.1, and Java 8u102. All experiments were repeated several times to remove outliers. To avoid caching effects, we executed a word count program between all executions of test scripts.

Figure 6 shows the execution time without profiling as well as with code instrumentation for profiling for each of our test scripts. It can be seen that profiling incurs only a small overhead for a sample rate of 1/1 and 1/2 (meaning 100% or 50% respectively are selected) and runtimes only differ in less than 10 s or 5%. For our other experiments we selected a sample rate of 10%.

Figure 7 shows runtimes for each script for three executions. Prior to the first execution no statistics were available and thus are collected during this execution (blue bars). In the second execution, previously generated statistics were used to decide which operator to materialize (orange bars). Hence, this execution includes also the writing of the intermediate result. For the last execution the materialized results were loaded (green bars) and thereby reduced the overall execution time of the job.

As we argued in previous sections, scripts are often developed incrementally. In this experiment we show the runtime differences for one script of each scenario for incremental execution (Fig. 8). We first ran the according script without materialization support (dashed orange lines) and compared the runtimes to an execution with materialization enabled (dotted blue lines). On the x-axis, step 0 is the initial execution with a LOAD and a first FILTER operator and subsequent steps add one or more operators. For the first few steps, both execution times are equal except for some minimal discrepancies. At some point however, the

⁶ <http://www1.nyc.gov/site/planning/data-maps/open-data.page>.

⁷ <https://www.gdelproject.org/data.html>.

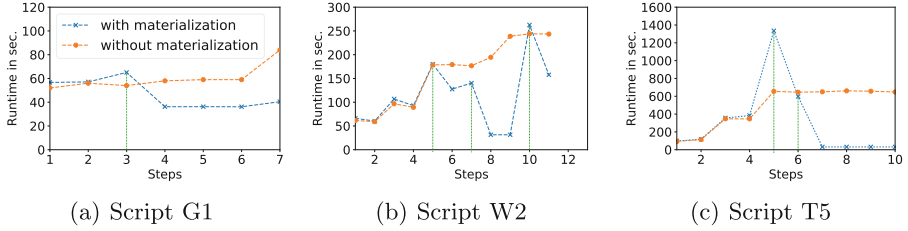


Fig. 8. Incremental execution of one script for each scenario (different y scales). (Color figure online)

optimizer recognizes a materialization point for which a benefit will be achieved and writes the respective result to disk (indicated by vertical lines). In this step, the execution time for *with materialization* rises above the reference time *without materialization*. However, since this is executed only once, the additional costs (clearly visible in Fig. 8(c)) will easily be amortized in subsequent executions that benefit from loading the materialized data. In fact, materialization reduced the cumulative execution time for G1 from 7 to 5 min, for W2 from 30 to 20 min, and for T5 from 80 to 30 min. The estimation of the benefit is an important aspect of our approach. In our experiments we saw that the estimated benefits often were close to real measured speedups, but sometimes also deviated to some extent. We observed that in almost all cases when a benefit could be achieved, we underestimated it – meaning that a subsequent execution was even shorter than calculated. In our tests we never encountered the situation that the optimizer calculated a benefit for a candidate materialization point which actually did not bring any benefit during execution. From this we conclude that our cost model calculates executions costs well enough to select an appropriate materialization point and avoid candidate materialization points that would cause longer execution times. Deviations are caused by our current implementation of the time measurement which depends on the information of a partition’s parent(s). If the parent partitions could not be determined precisely the computation of the respective operators execution time may assume a shorter or longer time.

To test the impact of the selected strategy (cf. Sect. 4.4) on the performance, we looked at two cases: In the first case we used additional scripts that all share the first six operators and then diverge into their individual paths that all contain another five operations. Strategy `last` did not materialize data as the last candidate materialization point of a job will not be repeated and thus no job benefited from recycling and execution for each script took around 420 s (7 min). For strategies `maxbenefit` and `markov` intermediate results were recycled and execution time was around 160 s (2:40 min) for both when results were loaded. In the second case we disassembled the jobs from our three use case scenarios into a total of 132 small jobs, executed them one after the other in a random sort order and captured the time it took to complete all jobs. For strategy `last` the execution time was 3:47 h, while for `maxbenefit` and `markov` the total time was 2:52 h and 2:48 h, respectively. This shows that a selection strategy that

takes the costs of operators into account achieves good results. In this setting `maxbenefit` and `markov` created similar results, but `markov` strategy performed slightly better in this last experiment as it sometimes chose other materialization point than `maxbenefit`, which more subsequent jobs could recycle.

7 Summary and Outlook

In this paper we presented an approach for a cost-based decision model to speed up execution of dataflow programs by merging jobs and reusing intermediate results accross multiple executions of the same or different jobs. The model was implemented in our Pig-to-Spark compiler Piglet which injects profiling code into the submitted jobs and rewrites them to materialize intermediate results or reuse existing results. In our evaluation we showed that profiling does not add any significant overhead to execution time, that jobs greatly benefit from reusing existing results, and that different strategies for choosing which intermediate result to materialize are needed.

In future work we will address the problem of cache replacement as there is no infinite space for storing the materialized results and existing data may need to be removed. Furthermore, our current implementation is based on the lineage information of operators and could be improved by implementing strategies for query containment checks. To address the fact that often machines are rented for data processing, the monetary costs of CPU cycles and storage may also be integrated into our cost model.

References

1. Abiteboul, S., Duschka, O.M.: Complexity of answering queries using materialized views. In: PODS, pp. 254–263 (1998)
2. Camacho-Rodrguez, et al.: PigReuse: A Reuse-based Optimizer for Pig Latin. Technical report, Inria Saclay (2016)
3. Chao-Qiang, H., et al.: RDDShare: reusing results of spark RDD. In: DSC, pp. 370–375 (2016)
4. Chirkova, R., Halevy, A.Y., Suci, D.: A formal perspective on the view selection problem. In: VLDB, pp. 59–68 (2001)
5. Elghandour, I., Abounaga, A.: Restore: reusing results of mapreduce jobs. In: VLDB, vol. 5, pp. 586–597 (2012)
6. Halevy, A.Y.: Answering queries using views: a survey. VLDB J. **10**(4), 270–294 (2001)
7. Harinarayan, V., Rajaraman, A., Ullman, J.D.: Implementing data cubes efficiently. SIGMOD Rec. **25**(2), 205–216 (1996)
8. Idreos, S., et al.: Merging what’s cracked, cracking what’s merged: adaptive indexing in main-memory column-stores. PVLDB **4**(9), 585–597 (2011)
9. Larson, P.Á., Yang, H.Z.: Computing queries from derived relations: theoretical foundation. University of Waterloo, Department of Computer Science (1987)
10. Nykiel, T., et al.: MRShare: sharing across multiple queries in MapReduce. PVLDB **3**(1–2), 494–505 (2010)

11. Perez, L.L., Jermaine, C.M.: History-aware query optimization with materialized intermediate views. In: ICDE, pp. 520–531. IEEE, March 2014
12. Sattler, K., Geist, I., Schallehn, E.: QUIET: continuous query-driven index tuning. In: VLDB, pp. 1129–1132 (2003)
13. Schnaitter, K., Abiteboul, S., Milo, T., Polyzotis, N.: COLT: continuous on-line tuning. In: SIGMOD, pp. 793–795 (2006)
14. Sparks, E.R., et al.: KeystoneML: optimizing pipelines for large-scale advanced analytics. In: ICDE, pp. 535–546 (2017)
15. Srivastava, D., Dar, S., Jagadish, H.V., Levy, A.Y.: Answering queries with aggregation using views. In: VLDB, vol. 96, pp. 318–329 (1996)
16. Valentin, G., et al.: DB2 advisor: an optimizer smart enough to recommend its own indexes. In: ICDE, pp. 101–110 (2000)
17. Wang, G., Chan, C.-Y.: Multi-query optimization in MapReduce framework. In: PVLDB, pp. 145–156 (2013)
18. Yang, H.Z., Larson, P.Å.: Query transformation for PSJ-queries. In: PVLDB, vol. 87, pp. 245–254 (1987)
19. Zhang, Y., Duc, P.M., Corcho, O., Calbimonte, J.-P.: SRBench: a streaming RDF/SPARQL benchmark. In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012. LNCS, vol. 7649, pp. 641–657. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35176-1_40



ATUN-HL: Auto Tuning of Hybrid Layouts Using Workload and Data Characteristics

Rana Faisal Munir^{1,2(✉)}, Alberto Abelló¹, Oscar Romero¹, Maik Thiele²,
and Wolfgang Lehner²

¹ Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
{[fmunir](mailto:fmunir@essi.upc.edu), [aabello](mailto:aabello@essi.upc.edu), [oromero](mailto:oromero@essi.upc.edu)}@essi.upc.edu

² Technische Universität Dresden (TUD), Dresden, Germany
{[maik.thiele](mailto:maik.thiele@tu-dresden.de), [wolfgang.lehner](mailto:wolfgang.lehner@tu-dresden.de)}@tu-dresden.de

Abstract. Ad-hoc analysis implies processing data in near real-time. Thus, raw data (i.e., neither normalized nor transformed) is typically dumped into a distributed engine, where it is generally stored into a hybrid layout. Hybrid layouts divide data into horizontal partitions and inside each partition, data are stored vertically. They keep statistics for each horizontal partition and also support encoding (i.e., dictionary) and compression to reduce the size of the data. Their built-in support for many ad-hoc operations (i.e., selection, projection, aggregation, etc.) makes hybrid layouts the best choice for most operations.

Horizontal partition and dictionary sizes of hybrid layouts are configurable and can directly impact the performance of analytical queries. Hence, their default configuration cannot be expected to be optimal for all scenarios. In this paper, we present ATUN-HL (Auto TUNing Hybrid Layouts), which based on a cost model and given the workload and the characteristics of data, finds the best values for these parameters. We prototyped ATUN-HL for Apache Parquet, which is an open source implementation of hybrid layouts in Hadoop Distributed File System, to show its effectiveness. Our experimental evaluation shows that ATUN-HL provides on average 85% of all the potential performance improvement, and 1.2x average speedup against default configuration.

Keywords: Big data · Hybrid storage layouts · Auto tuning · Parquet

1 Introduction

Data analysis plays a decisive role in today's data-driven organizations, which increasingly produce and store large volumes of data in the order of petabytes to zettabytes [16]. The storage and processing of such data has imposed a shift in the hardware, from single machines to large scale distributed systems. Apache Hadoop¹ is a pioneer large-scale distributed system and consists of a storage

¹ <https://hadoop.apache.org>.

layer, namely Hadoop Distributed File System (HDFS)², and a processing layer, namely MapReduce [6]. The former allows to keep data in raw format without any normalization or pre-processing. The latter allows data-intensive flows (DIFs) to process raw data such that they are ready for the analysis.

Hadoop and many modern in-memory processing engines (i.e., Apache Spark³) provide high-level languages (i.e., Apache Pig and Hive, SparkSQL) that facilitate writing DIFs for processing raw data (e.g., removing dirty data, integrating multiple data sources) stored in HDFS. Typically, the processed data is stored as a very wide table for analytical queries, because of its advantages over normalized tables [4, 12]. Hybrid layouts are de-facto preferred options for storing such wide tables, due to their built-in support for many basic operations (i.e., selection, projection, aggregation, etc.) allowing ad-hoc analysis, without the need of moving the data to other storage (i.e., relational, document store, etc.).

There are several available hybrid layout implementations, such as: Optimized Record Columnar (ORC)⁴, Parquet⁵ and CarbonData⁶. All of them follow the same physical structure. Data is stored into multiple horizontal partitions, known as stripes in ORC, row groups (RGs) in Parquet and blocklet in CarbonData, and each horizontal partition stores its data column-wise. Hybrid layouts also store min-max statistics [13] for each horizontal partition to help in filtering (i.e., partitions that do not match predicates of a query are skipped). In addition, they support dictionary encoding to encode repetitive values, that can also be used for further filtering partitions.

Table 1. Effect of horizontal partition size

	Small Partition	Large Partition
Parallelism	+	-
Task overhead	-	+
Filtering	+	-
Metadata size	-	+
Dictionary encoding	-	+
Memory buffering	+	-
Load balancing	+	-

Despite having default values, the sizes of horizontal partitions and dictionary are configurable, depending on the type of workload. Thus, their values should be decided based on the data characteristics and usage. For instance, it is recommended to have a small size of horizontal partition for low selectivity queries and, a large size for high selectivity queries. However, it is not straight-forward to find an optimal size for all the queries, because this depends on their concrete selectivity and the type of data they access, therefore the problem becomes challenging. Moreover, the size of horizontal partitions can also effect different

² https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

³ <https://spark.apache.org>.

⁴ <https://orc.apache.org>.

⁵ <https://parquet.apache.org>.

⁶ <https://carbodata.apache.org>.

execution settings, which is shown in Table 1. It can be seen that small partitions positively impacts parallelism (by increasing the number of parallel tasks), filtering (by skipping unmatched partitions using statistics), memory buffering (they require less memory to buffer the data before flushing to the disk), and load balancing (by better distributing the loads among multiple machines). Whereas, large horizontal partitions help positively to reduce task overhead (by reading less metadata and reducing Java garbage collector overhead), metadata size (by storing less statistics), and also helps in performing better encoding (by encoding large number of repetitive values). In this paper, we aim at improving filtering, metadata size and dictionary encoding by choosing the optimal partition size.

Similarly, the characteristics of data require different dictionary sizes to handle different attribute lengths and number of distinct values. The dictionary is not only important for compression, but it can also be used to filter partitions. Specifically, when data is unsorted and it is not possible to filter partitions simply using min-max statistics, as we will show in Sect. 5.3.

In this paper, we present our approach, namely ATUN-HL, which helps to find best values for the aforementioned parameters using a cost model, which estimates the optimal values for the size of the horizontal partition and the dictionary, based on the given workload and data characteristics. Moreover, it should also be noted that the chunk size of HDFS is always greater than or equal to the horizontal partition size. Hence, it should be configured accordingly. We instantiated ATUN-HL for Parquet, to show its applicability in real scenarios and conducted an extensive evaluation on TPC-H⁷ to show that ATUN-HL can significantly improve the query response times over Parquet with default configuration.

The main contributions of this work can be summarized as follows:

- We extend the cost model for hybrid layouts presented in [14].
- We propose ATUN-HL, a framework to optimize hybrid layouts.
- We prototype ATUN-HL on Parquet to show its benefits.
- We report the results of our extensive evaluation with TPC-H benchmark.

The remaining paper is organized as follows. In Sect. 2, we discuss the related work. In Sects. 3 and 4, we discuss the cost model and our approach in detail. In Sect. 5, we show our experimental results. Finally, in Sect. 6, we conclude the paper.

2 Related Work

In [7], an indexing technique is proposed for hybrid layouts. The indexing information is stored as metadata per RG and data node, which in turn enables filtering RGs in selective queries. However, this approach uses default RG size, which as previously argued does not always perform well.

In [17, 18], different partitioning approaches are presented, which help in selective queries. In [17], data is divided into multiple horizontal partitions and

⁷ <http://www.tpc.org/tpch>.

in each partition, data is stored row-wise, rather than column-wise. It also stores extra meta information for each partition, which is computed based on the predicates of queries. Predicates are used as features, where a bit is stored for each tuple matching a feature. This eventually gives a feature-vector for every tuple, which is then used for filtering partitions. A similar vector is also used in [18], however this time it utilizes hybrid layouts with column grouping, instead of fixed row layouts. The latter helps for both selection and projection queries. Yet, these techniques fall short when it comes to tuning the configurable parameters and they only provide new strategies of partitioning.

In [4], a column reordering technique is proposed to reduce the disk seek cost for hybrid layouts by storing together the columns which are accessed by the same queries. In addition, this approach sometimes duplicates columns to store them in a contiguous way. It helps to reduce the disk seek cost and overall improves the query execution time. However, it still does not try to find the optimal configuration values of hybrid layouts based on the running workload.

There are other works [2, 3, 11, 15], which try to use different layouts based on the workload. The goal is always to store the data in the most appropriate one for the given workload. Yet again, they do not optimize the layouts.

There are still few research works [9, 10] available on tuning big data analytical platforms (such as Hadoop). They focus on finding the optimal values for each configuration parameter available in a big data analytical system. Nevertheless, they target overall systems rather than individual layouts. These techniques can be used as complementary to our approach.

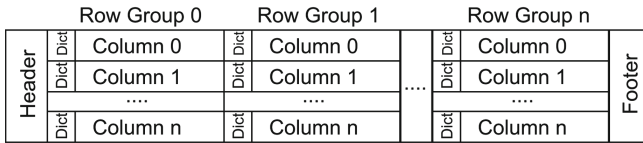


Fig. 1. Physical structure of hybrid layouts

3 Cost Model

In this section, we extend a cost model of our previous work [14]. Specifically, we refine the selection cost model based on the use of min-max statistics and dictionary encoding. Further, we extend our cost model to estimate the dictionary size for hybrid layouts.

First, we present the physical structure of hybrid layouts, which helps to build the cost model. Based on that, we estimate the cost of selections and the size of the dictionary. The former helps to find the optimal RG size. Our cost model considers two scenarios to estimate the selection cost, which are as follows: filtering using min-max statistics and using the dictionary. Likewise, it considers two types of dictionaries, i.e., global and local.

Table 2. Parameters of the cost model

Variable	Description
<i>System constants</i>	
p	Probability of accessed replica being local
$Chunk_{Size}$	Disk assignment unit size in HDFS
BW_{Disk}	Disk bandwidth
BW_{Net}	Network bandwidth
$Time_{Seek}$	Disk seek time
$Time_{Disk}$	$Chunk_{Size}/BW_{Disk}$
$Time_{Net}$	$Chunk_{Size}/BW_{Net}$
<i>Data statistics</i>	
$ T $	Number of rows in a table
$ColValue_{Size}$ ^a	Average size of a column value
$\#Cols$	Total columns of T
$ C $	Distinct values of a column
$ D $	Number of values in the dictionary
$Sorted_{Col}$	True for sorted and False for unsorted data
<i>Workload statistics</i>	
SF	Selectivity factor of a query
<i>Hybrid layouts variables</i>	
RG_{Size}	Row group size
$MetaRG_{Size}$	Size of meta data for an RG
$Marker_{Size}$	Size of sync marker

^aExtra 4 bytes are considered for variable length columns

As shown in Fig. 1, the data is divided into RGs (i.e., horizontal partitions), and inside each RG, it is stored column-wise. Further, if dictionary encoding is possible, first dictionaries are stored per column and afterwards the corresponding encoded data. If dictionary encoding is not possible, then the data values are stored contiguously without any encoding. Moreover, hybrid layouts also store metadata (e.g., min-max statistics) for each RG inside either the header or footer section. Thus, the size of hybrid layouts depends on the size of the actual data and metadata.

Our cost model for hybrid layouts relies on a wide range of statistical information that are summarized in Table 2, containing system constants, data statistics, workload statistics as well as hybrid layout variables. We assume that the constants which depend on the configuration of the environment (e.g., BW_{Disk} , BW_{Net}) are provided. Furthermore, we discuss the collection of statistics (e.g., dataset and workload) in Sect. 4.

$$Used_{RowGroups} = \frac{(ColValue_{Size} \cdot |T| + Marker_{Size}) \cdot \#Cols}{RG_{Size}} \quad (1)$$

$$|RG| = \frac{|T|}{Used_{RowGroups}} \quad (2)$$

$$TotalMetaSize = (Meta_{RGSize} \cdot \#Cols) \cdot Used_{RowGroups} \quad (3)$$

3.1 Estimating the Selection Cost

The selection cost model estimates the number of RGs read from the disk and as well as the total read size. For this, first we need to estimate the total number of RGs using Eq. 1, and the number of rows in an RG ($|RG|$) using Eq. 2. Further, we also need to estimate the total size of metadata (cf. in Eq. 3), which is always read from disk to check the matching RGs. Our selection cost model focuses on two cases as discussed earlier. The first one considers filtering using min-max statistics of each RG, and second one filtering using the dictionary.

$$Read_{RowGroups} = \begin{cases} SF \cdot Used_{RowGroups} + 1 & \text{sorted data} \\ Used_{RowGroups} & \text{unsorted and min-max} \\ (1 - (1 - SF)^{|RG|}) \cdot Used_{RowGroups} & \text{unsorted and dictionary} \end{cases} \quad (4)$$

Filtering Using Min-Max Statistics. There are two extreme cases when hybrid layouts use min-max statistics to filter RGs, depending on whether data is sorted or not. If data is completely sorted then the selected data will always be contiguous and we can calculate the total number of read RGs based on the selectivity factor as shown in Eq. 4. We add one to handle the effect of position variation inside the RGs for sorted data, because hybrid layouts read the whole RG even if there is only one matching row. The reason to add one is illustrated in Fig. 2. It shows two RGs and each has 5 rows. Let us assume that we select 3 rows. There are two possible scenarios: (A) there is no overlap and only one RG is read from disk; and (B) there is an overlap and two RGs are read. If we take the average of all possible positions of the first selected row in the first RG, it gives approximately $(SF \cdot Used_{RowGroups}) + 1$.

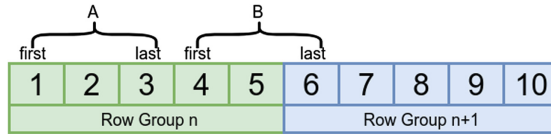


Fig. 2. Effect of position variation inside the RGs

If data is completely unsorted (i.e., uniform distribution), it is unlikely (shown in Sect. 5.3) to skip any RG, because the distribution of data makes the min-max range of each RG too wide. Hence, the read RGs will be the same as the total number of RGs. We will also experimentally show in Sect. 5.3 the ineffectiveness of min-max statistics for uniformly distributed unsorted data. Intermediate cases exist for different kinds of skewness, and Eq. 4 could be enriched with corresponding estimations without affecting the rest of the paper.

Filtering Using the Dictionary. The dictionary can also be used to filter RGs when data is encoded. When min-max statistics fail to filter any RG, the dictionary is still very useful, because it contains all existing values. The number of RGs required to be read from disk can be estimated as in Eq. 4 (borrowed from bitmap indexes [5]).

$$Used_{Chunks} = \left\lceil \frac{Used_{RowGroups} \cdot RG_{Size}}{Chunk_{Size}} \right\rceil \quad (5)$$

$$Read_{Size} = (Read_{RowGroups} \cdot RG_{Size}) + (TotalMeta_{Size} \cdot Used_{chunks}) \quad (6)$$

$$|Chunk| = \left\lfloor \frac{Chunk_{Size}}{RG_{Size}} \right\rfloor \quad (7)$$

$$Chunk_{Seeks} = \begin{cases} \frac{Read_{RowGroups}}{|Chunk|} + 1 & \text{if sorted} \\ Used_{Chunks} \cdot \left(1 - \left(1 - \frac{Read_{RowGroups}}{Used_{RowGroups}}\right)^{|Chunk|}\right) & \text{if unsorted} \end{cases} \quad (8)$$

$$W_{ReadTransfer} = \frac{Time_{Disk} + (1-p) \cdot Time_{Net}}{Time_{Seek} + Time_{Disk} + (1-p) \cdot Time_{Net}} \quad (9)$$

$$Query_{Cost} = \frac{Read_{Size}}{Chunk_{Size}} \cdot W_{ReadTransfer} + (Chunk_{Seeks} + Used_{Chunks}) \cdot (1 - W_{ReadTransfer}) \quad (10)$$

The above equations give the expected number of RGs being read from disk, which helps in estimating the total query cost. In distributed processing engines, the data is processed in multiple tasks in parallel and the number of tasks equals to the number of chunks used to store the data, which can be estimated using Eq. 5.

Moreover, we observed that each task reads all the metadata separately. The reason is that the distributed processing engines (such as Hadoop and Spark) create a separate process for each task with its own memory. This memory is not accessible to other tasks and hence, forces to read all metadata, and consequently, increases the reading size. We consider this in Eq. 6, where we estimate the total read size.

Additionally, we take into consideration the disk seek cost, which depends on the number of chunks being read and also on the number of seeks required to fetch the metadata. The former is equal to the number of read chunks if data is sorted, because it reads consecutive RGs. In Eq. 7, we calculate the total number of RGs inside a chunk, which is used in Eq. 8 to estimate the total number of seeks for sorted data. Similar to filtering, we add add one to Eq. 7 to handle the effect of position variation of RGs inside chunks. On the other hand, when data is unsorted, number of seeks is directly influenced by the distribution of the read RGs, which are non-consecutive due to fact that any RG can match the predicate independently of its position. Thus, it can be approximated by estimating how many RGs are read from a chunk, which depends on the total number of RGs inside a chunk, again calculated using Eq. 7. Similarly, we need to estimate the total seeks for reading metadata. As discussed earlier, typically, metadata is stored in the header or footer sections and one seek is required to

locate it on the disk. Additionally, it is always read separately in every task, hence the total seeks of metadata will be equal to the total number of tasks (which is equal to the number of chunks).

In distributed processing engines, sometimes, they require to read the data remotely (for instance, it depends on occupancy of machines and unbalanced distribution of workload) and for it, we use a probability p to indicate the likelihood of chunks being accessed locally (i.e., data shipping through the network is needed to reach the operation executor). This is used in Eq. 9 to estimate the weight (to calculate the resources usage) of transferring the chunk data compared to the corresponding seek time. Further, it is used along with the total number of seeks in Eq. 10 to estimate the total query cost.

$$|D| = \begin{cases} |C| & \text{for global dictionary} \\ \lceil |C| \cdot (1 - ((|C| - 1)/|C|)^{|RG|}) \rceil & \text{for local dictionary} \end{cases} \quad (11)$$

$$DictionarySize = |D| \cdot ColValueSize \quad (12)$$

$$UsedBits = \lceil \log_2 |D| \rceil \quad (13)$$

$$EncodedColSize = \begin{cases} UsedBits \cdot |T| & \text{for global dictionary} \\ UsedBits \cdot |RG| & \text{for local dictionary} \end{cases} \quad (14)$$

3.2 Estimating the Size of the Dictionary

As discussed earlier, hybrid layouts support dictionary encoding, which helps to encode repetitive values to reduce the size and also to facilitate filtering RGs. There are different implementations of dictionary encoding in different types of hybrid layouts. For instance, CarbonData uses a global dictionary to encode the data, whereas Parquet uses a local dictionary inside every RG. However, these two implementations can be easily handled by the same cost model.

Global Dictionary. The size of the dictionary depends on the number of values to store inside, which is the total distinct values (i.e., $|C|$) of a column estimated in Eq. 11. The size of the dictionary for one column can be then estimated using Eq. 12. Further, the average number of bits required to encode one value are estimated in Eq. 13, and used in Eq. 14 to estimate the encoded size of the data.

Local Dictionary. Similarly, the size of the local dictionary depends on the number of values to be put inside the dictionary of an RG, which is the same as the distinct values of a column inside an RG. We estimate the total number of expected distinct values⁸ inside an RG as shown in Eq. 11. Next, similar to global dictionary, the average number of bits required to encode one value are estimated in Eq. 13, and used further in Eq. 14 to estimate the encoded size of the data.

⁸ <https://math.stackexchange.com/questions/72223/finding-expected-number-of-distinct-values-selected-from-a-set-of-integers>.

4 ATUN-HL

In this section, we first discuss about the collection of data and workload characteristics. Next, we explain our methodology, which utilizes the cost model to find the optimal sizes for RG and dictionary.

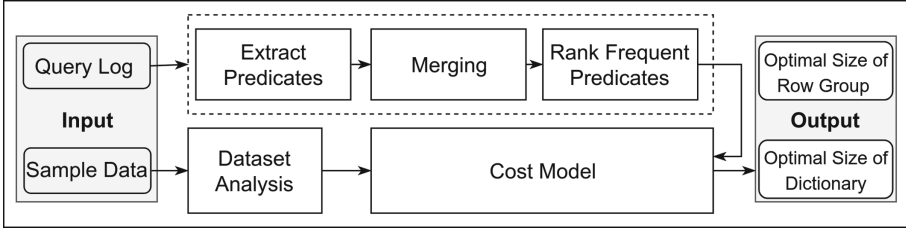


Fig. 3. Overview of ATUN-HL

4.1 Collecting Workload and Data Characteristics

Figure 3 shows the overview of our approach. It takes a query log and the sample data as input, and analyzes them in different components to extract statistical information. The query log is used to extract the information related to the workload. First, our approach extracts the clauses from all the query representatives. Second, it merges the similar clauses or the clauses that can be subsumed. Thirdly, it applies frequent itemset mining approach [8], to rank the most frequent clauses. Finally, it takes the top-k clauses to extract the workload information to be considered. On the other hand, dataset analysis module takes a sample of data and computes the statistical information listed in Table 2. We use the single column profiling technique from [1].

The use of query log to optimize the parameters for future workloads is justified in [17, 18], which conclude that filters are recurring and only a small portion are entirely new over time.

4.2 Finding the Best Configuration Parameters

Let us assume T is a wide table and has a set of columns defined by $C = \{c_1, c_2, \dots, c_n\}$. Similarly, a workload is defined as $Q = \{q_1, q_2, \dots, q_n\}$, the frequent clauses extracted from Q are defined as $P = \{p_1, p_2, \dots, p_n\}$ the total cost of workload is calculated as $Cost_P(RG_{Size}, Z) = \sum_{p \in P} QueryCost(RG_{Size}, Z)$, where Z represents the total size of T (considering dictionary encoding if needed). Our goal is to minimize $Cost_P$ by selecting the best RG and dictionary sizes.

Algorithm 1 shows the steps to find the optimal sizes of RG and dictionary. It initializes a set in line 1 with the element 0, which corresponds to the scenario where dictionary encoding is completely disabled for all columns. Next, in lines 2 to 4, it iterates over all the columns, computes their dictionary sizes, rounds

Algorithm 1. Finding the best size of RG and dictionary

```

1  $Possible_{Dict_{Sizes}} = \{0\}$ ;
2 for  $c \in Cols$  do
3    $Dict_{Size} = RoundUpToKiloBytes(EstimateDictionarySize(c))$ ;
4    $Possible_{Dict_{Sizes}}.insert(Dict_{Size})$ ;
5 end
6  $Best = [\infty, 0, 0]$ ; //  $Best[Cost, RG_{Size}, Dict_{Size}]$ 
7 for  $Dict_{Size} \in Possible_{Dict_{Sizes}}$  do
8    $Z = EstimateEncodedSize(Dict_{Size})$ ;
9    $Curr_{RG_{Size}} = Solver(\frac{d}{dRG_{Size}}(Cost_P(RG_{Size}, Z)) = 0)$ ;
10   $Curr_{Cost} = Cost_P(Curr_{RG_{Size}}, Z)$ ;
11  if  $Curr_{Cost} < Best.Cost$  then
12     $Best = [Curr_{Cost}, Curr_{RG_{Size}}, Dict_{Size}]$ ;
13  end
14 end
15 return  $Best$ ;

```

them up to the nearest kilobytes, and stores them inside the set. Further, in lines 7 to 12, it iterates over all those dictionary sizes and computes the table size according to the current processed dictionary size. Then, the encoded size is used to find the optimal RG size by solving the derivative of the overall cost function. Finally, this value is used to compute the corresponding cost. If the cost is smaller than the best until now, we keep the current processed dictionary and RG sizes as the best ones.

In order to be able to find the minimum cost, we derive the function with respect to the RG size (i.e., $\frac{d}{dRG_{Size}}(Cost_P(RG_{Size}, Z)) = 0$). Equation 15 shows the overall query cost after replacing all variables except read RGs, which still depends on how data has been stored (see Eq. 4). Notice that, we need to remove the ceiling function of Eq. 5, as well as floor from Eq. 7. We can do the former, because the number of chunks is much smaller than the total number of RGs, and it is only used in calculating the meta size and seek cost, and both are very small compared to the total reading size. Similarly, we can also remove floor in Eq. 7, due to its negligible impact on overall cost. We validated their removal with detailed experiments (see Sect. 5.3).

$$\begin{aligned}
 Z &= \begin{cases} (ColValue_{Size} \cdot |T| + Marker_{Size}) \cdot \#Cols & \text{no encoding} \\ (Dictionary_{Size} + EncodedCol_{Size} + Marker_{Size}) \cdot \#Cols & \text{encoding} \end{cases} \\
 Y &= Meta_{RG_{Size}} \cdot \#Cols \\
 Query_{Cost}(RG_{Size}, Z) &= \frac{ReadRowGroups \cdot RG_{Size} + \frac{Y \cdot Z^2}{RG_{Size} \cdot Chunk_{Size}}}{Chunk_{Size}} \quad (15)
 \end{aligned}$$

$$\cdot W_{ReadTransfer} + \frac{ReadRowGroups + \frac{Z}{RG_{Size}}}{\frac{ChunkSize}{RG_{Size}}} \cdot (1 - W_{ReadTransfer})$$

5 Experimental Results

In this section, we discuss the setup and the dataset used for our experiments. We also show the ineffectiveness of min-max statistics and usefulness of dictionary for unsorted data. Moreover, we provide the results to validate the accuracy of the cost model and to show the benefits of our approach.

Table 3. Values according to our environment

Variable	Value
p	0.97
$ChunkSize$	512 MB
BW_{Disk}	1.3×10^8 bytes/s
BW_{Net}	1.25×10^8 bytes/s
$Time_{Seek}$	5.0×10^{-3} s
$Meta_{RG_{Size}}$	156 bytes
$Marker_{Size}$	16 bytes

5.1 Setup

The machine used in our evaluation has a Xeon E5-2630L v2 @2.40 GHz CPU, 128 GB of main memory, and 1TB SATA-3 of hard disk, and runs Hadoop 2.6.2 and Spark 2.1.10 on Ubuntu 14.04 (64 bit). Our approach is evaluated under two settings: a single node and a 4-machines cluster⁹. In the cluster, we dedicated one machine to HDFS name node and Spark master node together, and the remaining three machines to data nodes for Hadoop and workers for Spark.

We prototyped our approach for Apache Parquet 1.8.2, which further divides each column into multiple data pages (i.e., 1 MB) and also stores min-max statistics per data page (i.e., 53 bytes). Nevertheless, currently Parquet does not support data page filtering, so we applied the cost model as described above. If needed, our cost model could be easily adapted to data page filtering by simply replacing RG size with data page size and $|RG|$ with the number of rows of a data page.

⁹ <http://www.ac.upc.edu/serveis-tic/altas-prestaciones>.

Table 3 shows the values of all environmental variables in our testbed. In addition, default RG and dictionary sizes in Parquet are 128 MB and 1 MB, which we use in our evaluation together with best and worse obtained costs.

5.2 Dataset

As mentioned in [4, 12], very wide tables are common in modern analytical systems, because of their advantages in processing compared to normalizing data into narrower tables. Nevertheless, in TPC-H, the widest table has only 16 columns and in TPC-DS¹⁰, only 26. To the best of our knowledge, there is no public benchmark available that consists of wide tables. Hence, we follow [17] to generate a wide table by completely denormalizing all other tables in TPC-H against *lineitem*. The FROM clauses in all queries are consequently changed to the corresponding denormalized table.

5.3 Results

We perform four types of evaluations for our approach. Firstly, we show the drawbacks of min-max based filtering for unsorted data through statistical and also experimental evaluation. Secondly, we show the benefits of dictionary based filtering for unsorted data. Thirdly, we validate the accuracy of our cost model. Finally, we show the performance improvements of our approach on the cluster by comparing it to the baseline setting.

Usefulness of Min-Max Statistics. As previously discussed, min-max statistics are not useful for unsorted data, because uniform data distribution makes it impossible to skip RGs. This behavior is validated with a detailed statistical and experimental evaluation.

$$P_{\text{Skipping}} = \frac{\sum_{i=1}^{|C|} \left(\left(\frac{i-1}{|C|} \right)^{|RG|} + \left(\frac{|C|-i}{|C|} \right)^{|RG|} \right)}{|C|} \quad (16)$$

$$Read_{\text{RowGroups}} = (1 - P_{\text{Skipping}}) \times Used_{\text{RowGroups}} \quad (17)$$

Since point queries (i.e., those that search one single value) have higher probability of skipping an RG than the other supported types (namely interval and list of values), and also because of space limitation, we only provide a statistical cost model for this in Eq. 16. This estimates the probability of being outside of an RG, which would be the case if the value is less than the minimum of the RG or greater than the maximum. Thus, our cost model adds the probability of both (i.e., minimum and maximum) for each value of that column. Further, the probability of skipping one RG is used in Eq. 17, to find the total number of RGs read.

¹⁰ <http://www.tpc.org/tpcds>.

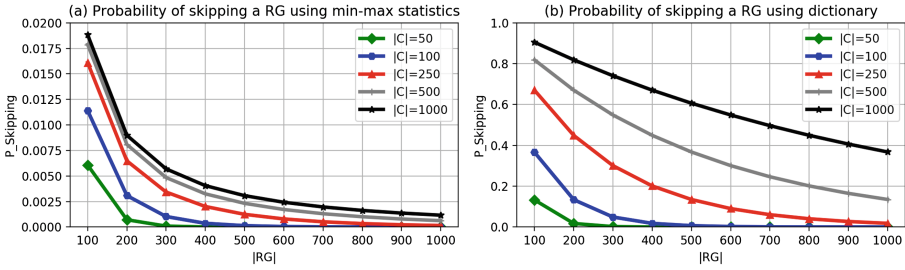


Fig. 4. Probability of skipping one RG

Figure 4a plots Eq. 16 for different number of rows $|RG|$, and different number of distinct values of a column $|C|$, which was confirmed with the corresponding experiments. We took 100 as the minimum for $|RG|$, because Parquet does not allow less rows per RG than that. Thus, it can be observed that the probability of skipping an RG is very low (i.e., always less than $<2\%$), confirming that min-max statistics are useless for unsorted data. Moreover, when the number of rows in an RG increases, the probability of skipping decreases, which means that it is almost certain that a full scan will be performed. A higher number of distinct values slightly increase the chances of skipping an RG, but it is still very unlikely for RGs with many rows.

Benefits of Dictionary Encoding. We also plot Eq. 4 for dictionary encoding (see Fig. 4b), confirming its superiority over min-max statistics. It can be seen that this clearly gives higher probability of skipping, but the chances of skipping still decrease quickly as the number of rows in an RG grows. Yet, it helps with low selectivity queries (when min-max statistics still fail).

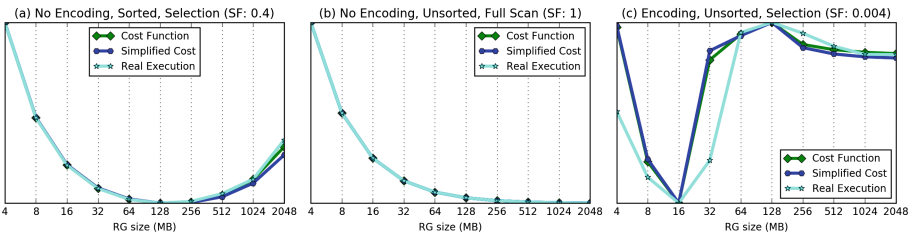


Fig. 5. Comparison between cost model, simplified version, and real execution

Cost Model Validation. Figure 5 shows the comparison of our cost model, the estimation through its simplified version (which allows derivation as presented in Eq. 15), and also actual execution (averaging 250 random runs). We normalized them $((x - min)/(max - min))$ to facilitate visual comparison. Moreover, as we will show below, the different units (as our cost model only considers I/O cost)

do not affect the quality of our prediction to choose the optimal RG size, since the estimated values always preserve the shape of the actual ones (i.e., minimum real cost is obtained for approximately the same value in the model).

We empirically validated the estimations on both sorted and unsorted data, with and without encoding. It can be seen that our cost model and its simplified version are very close and result in approximately the same value. Hence, the derivative can be safely used to find the optimal RG size. Moreover, these both versions follow exactly the same trend as the actual execution.

Performance Evaluation. We analyzed TPC-H queries to extract the clauses and ranked them according to their usage. The top 6 clauses which appear in 82% of the queries, are used to find the optimal RG and dictionary sizes. ATUN-HL chooses 30.76 MB (that we round up to 32 MB) for RG and 1 MB for dictionary.

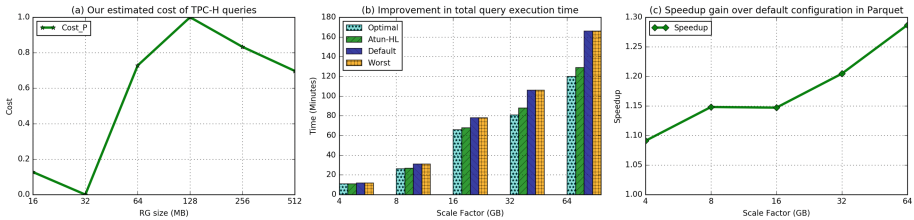


Fig. 6. Speedup gain

Figure 6a shows our estimated overall cost for TPC-H queries. It can be seen that ATUN-HL predicts the default RG size (i.e., 128 MB) as the worst configuration (being the minimum at 32 MB). As discussed earlier, it is very unlikely for Parquet to skip any RG, when the number of rows in an RG grows. When this turning point is crossed, the larger the RG the better, and our estimated cost depicts this behavior after 128 MB. Moreover, we also verified our estimation with detailed experiments as shown in Fig. 6b and c. Figure 6b compares the time improvements of ATUN-HL against the optimal, default, and worst configurations. ATUN-HL is not far from the optimal configuration, resulting in an 85% of all potential gain. Additionally, Fig. 6c shows the relative gain with regard to default RG size, which is 1.2X speedup on average (for the tested scale factors), clearly increasing with the increase in scale factor.

Finally, in Fig. 7, we also scrutinize the effect on individual query execution time for scale factor 64 GB. This shows that our approach improves the execution time of most of the queries, but does not help those actually performing a full scan (i.e., Q1, Q13, Q15, and Q16) because of one reason (i.e., high SF, >10%) or another (i.e., string matching using regular expression, which is not yet supported by Parquet). As shown above, the large RG size is always better for full scan.

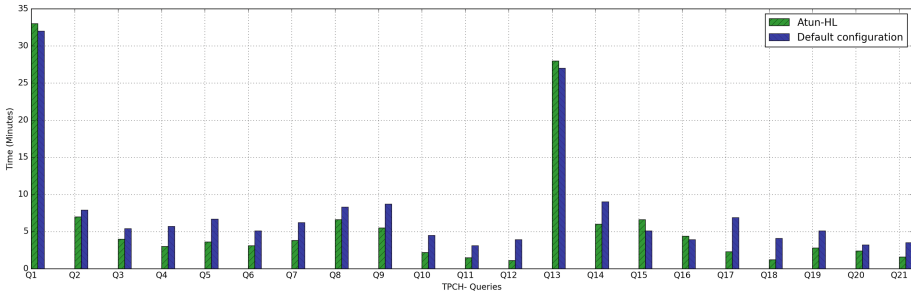


Fig. 7. Improvement in query execution time for 64 GB scale factor

6 Conclusions

Hybrid layouts are widely used to store processed data in highly distributed Big Data systems to perform ad-hoc analysis. Nevertheless, they have many configurable parameters that need to be tuned according to the characteristics of the data and workload, which can heavily impact query performance. Consequently, we proposed a cost-based approach to help optimizing such hybrid layouts. We prototyped our approach for Apache Parquet, evaluated it on TPC-H queries, and showed the improvement it provides.

Acknowledgement. This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate “Information Technologies for Business Intelligence - Doctoral College” (IT4BI-DC), and the GENESIS project, funded by the Spanish Ministerio de Ciencia e Innovación under project TIN2016-79269-R.

References

1. Abedjan, Z., Golab, L., Naumann, F.L.: Data profiling: a tutorial. In: SIGMOD Conference. ACM (2017)
2. Alagiannis, I., Idreos, S., Ailamaki, A.: H2O: a hands-free adaptive store. In: SIGMOD Conference. ACM (2014)
3. Azim, T., Karpathiotakis, M., Ailamaki, A.: Recache: reactive caching for fast analytics over heterogeneous data. PVLDB **11**(3), 324–337 (2017)
4. Bian, H., et al.: Wide table layout optimization based on column ordering and duplication. In: SIGMOD Conference. ACM (2017)
5. Cardenas, A.F.: Analysis and performance of inverted data base structures. Commun. ACM **18**(5), 253–263 (1975)
6. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
7. Ferreira, M., Paiva, J., Bravo, M., Rodrigues, L.E.T.: Smartfetch: efficient support for selective queries. In: CloudCom. IEEE Computer Society (2015)
8. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data Min. Knowl. Discov. **15**(1), 55–86 (2007)
9. Herodotou, H., Babu, S.: Profiling, what-if analysis, and cost-based optimization of mapreduce programs. PVLDB **4**(11), 1111–1122 (2011)

10. Herodotou, H., et al.: Starfish: a self-tuning system for big data analytics. In: CIDR (2011)
11. Jindal, A., Quiané-Ruiz, J., Dittrich, J.: Trojan data layouts: right shoes for a running elephant. In: SoCC. ACM (2011)
12. Li, Y., Patel, J.M.: Widetable: an accelerator for analytical data processing. PVLDB **7**(10), 907–918 (2014)
13. Moerkotte, G.: Small materialized aggregates: a light weight index structure for data warehousing. In: VLDB, pp. 476–487 (1998)
14. Munir, R.F., Abelló, A., Romero, O., Thiele, M., Lehner, W.: A cost-based storage format selector for materialization in big data frameworks. CoRR, abs/1806.03901 (2018)
15. Munir, R.F., Romero, O., Abelló, A., Bilalli, B., Thiele, M., Lehner, W.: ResilientStore: a heuristic-based data format selector for intermediate results. In: Bellatreche, L., Pastor, Ó., Almendros Jiménez, J.M., Aït-Ameur, Y. (eds.) MEDI 2016. LNCS, vol. 9893, pp. 42–56. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45547-1_4
16. Shvachko, K.V.: HDFS scalability: the limits to growth. *Login* **35**(2), 6–16 (2010)
17. Sun, L., Franklin, M.J., Krishnan, S., Xin, R.S.: Fine-grained partitioning for aggressive data skipping. In: SIGMOD Conference. ACM (2014)
18. Sun, L., Franklin, M.J., Wang, J., Wu, E.: Skipping-oriented partitioning for columnar layouts. PVLDB **10**(4), 421–432 (2016)



Set Similarity Joins with Complex Expressions on Distributed Platforms

Diego Junior do Carmo Oliveira¹, Felipe Ferreira Borges¹,
Leonardo Andrade Ribeiro^{1(✉)}, and Alfredo Cuzzocrea²

¹ Instituto de Informática, Universidade Federal de Goiás, Goiânia, Brazil
{diegooliveira,felipeferreiraborges,laribeiro}@inf.ufg.br

² DIA Department, University of Trieste and ICAR-CNR, Trieste, Italy
alfredo.cuzzocrea@dia.units.it

Abstract. A set similarity join finds all similar pairs from a collection of sets. This operation is essential for many important tasks in Big Data analytics including string data integration and cleaning. The vast majority of set similarity join algorithms proposed so far considers string data represented by a single set over which a simple similarity predicate is defined. However, real data is typically multi-attribute and, thus, better represented by multiple sets. Such a representation requires complex expressions to capture a given notion of similarity. Moreover, similarity join processing under this new formulation is clearly more expensive, which calls for distributed algorithms to deal with large datasets. In this paper, we present a distributed algorithm for set similarity joins with complex similarity expressions. Our approach supports complex Boolean expressions over multiple predicates. We propose a simple, but effective data partitioning strategy to reduce both communication and computation costs. We have implemented our algorithm in Spark, a popular distributed data processing engine. Experimental results show that the proposed approach is efficient and scalable.

Keywords: Similarity join · Distributed algorithms · Data integration

1 Introduction

The current Big Data era is characterized by large and rapidly growing datasets. This trend has fueled a substantial expansion of data analytics to reveal novel insights and guide business decisions. However, effectively deriving value from such datasets demands computationally intensive data preparation tasks [8]. For example, when data is collected from independent sources storing overlapping information, it invariably results in multiple representations of the same object which are not exact copies of one another. Such erroneous duplication affects statistics and can lead to inaccurate analysis results. Therefore, integrated datasets need to be cleaned before applying data mining algorithms.

Table 1. Records containing demographic information.

ID	Name	Street name	City
1	Susan Allen	William St.	New York
2	S. Allen	William St.	New York
3	Susan Allen	Woodley Ave.	Los Angeles
4	S. Willians	Allen St.	New York

Set similarity join is an essential operation for integration and cleaning of string data, which has attracted increasing attention over the years [1, 3, 6, 10–14, 17, 18]. After mapping strings to sets, we can employ a set similarity join to find all pairs of sets whose similarity is not less than a specified threshold. The underlying notion of similarity is captured by a *set similarity function* used in the join predicate. Set similarity join is attractive owing to its efficiency in dealing with string data, whose representation is typically sparse and high-dimensional, as well as its versatility in supporting a variety of similarity functions.

The vast majority of set similarity join algorithms proposed so far considers string data represented by a single set over which a simple similarity predicate is defined. However, real-world data is often multi-attribute. At first glance, traditional algorithms can still be used by simply selecting a single attribute for similarity assessment or concatenating string values from multiple attributes. However, both approaches may produce unsatisfactory results. For example, consider the records shown in Table 1. The four records actually represent three distinct individuals, because records 1 and 2 refer to the same person. If the similarity predicate is applied on the **Name** attribute alone, the pair formed by records 1 and 3 would be returned, whereas records 1 and 2 might not be considered similar owing to the abbreviation in the latter. If all attributes are concatenated instead, the pair formed by records 2 and 4 could be returned, because the value of **Name** in one record is similar to the value of **Street** in the other.

A better approach would be to represent multi-attribute data by multiple sets. Further, different similarity functions and thresholds can be defined, enabling the composition of more sophisticated similarity expressions. The main challenge of this approach is to incorporate such new formulation into a similarity join algorithm without hurting performance and scalability. Multi-attribute data is naturally larger and the comparison of records based on complex similarity expressions is computationally more expensive.

This paper presents a distributed set similarity join algorithm on multi-attribute data. To the best of our knowledge, this type of similarity join has been hardly investigated in a distributed environment. Our approach generalizes existing algorithms to enable similarity conditions based on complex Boolean expressions. We propose a simple, but effective data partitioning strategy to reduce both communication and computation costs. We have implemented our algorithm in Apache Spark, a popular distributed data processing engine [19].

Our experimental results show that the proposed solution is efficient and scalable when applied to large volumes of data.

The remainder of this paper is organized as follows. We provide background material in Sect. 2. Our proposed solution is presented in Sect. 3 and evaluated in Sect. 4. We discuss relevant related work in Sect. 5 before we wrap with the conclusions and overview of future work in Sect. 6.

2 Background

In this section, we first review basic concepts before formally define the problem. Finally, we provide a brief overview of the Spark platform.

2.1 Basic Concepts

Strings can be mapped to *sets of tokens* in several ways. For example, the string “*set similarity join*” can be mapped to the set of words $\{\text{‘set’}, \text{‘similarity’}, \text{‘join’}\}$. Another well-known method is based on the concept of *q-grams*, i.e., sub-strings of length q obtained by “sliding” a window over the characters of the input string. For example, the string “*similarity*” can be mapped to the set of 3-grams tokens $\{\text{‘sim’}, \text{‘imi’}, \text{‘mil’}, \text{‘ila’}, \text{‘lar’}, \text{‘ari’}, \text{‘rit’}, \text{‘ity’}\}$. In the following, we assume that all strings in the database have already been mapped to sets of tokens.

Given two sets x and y , a set similarity function $sim(x, y)$, returns a value in $[0, 1]$ to represent their similarity; larger value indicates that r and s have higher similarity.

Definition 1 (Set Similarity Functions). *Let x and y be two sets. Popular set similarity functions are defined as follows.*

- *Jaccard similarity:* $J(x, y) = \frac{|x \cap y|}{|x \cup y|}$.
- *Dice similarity:* $D(x, y) = \frac{2 \times |x \cap y|}{|x| + |y|}$.
- *Cosine similarity:* $C(x, y) = \frac{|x \cap y|}{\sqrt{|x| \times |y|}}$.

Whenever clear from context or unimportant to the discussion, we use the term similarity function (join) to mean set similarity function (join). Further, we focus henceforth on the Jaccard similarity, but all concepts and techniques presented in the following can be extended to Dice and Cosine as well [10].

Example 1. Consider the two token sets x and y below, which were derived from the strings “*similarity*” and “*similarigy*”.

$$x = \{\text{‘sim’}, \text{‘imi’}, \text{‘mil’}, \text{‘ila’}, \text{‘lar’}, \text{‘ari’}, \text{‘rit’}, \text{‘ity’}\},$$

$$y = \{\text{‘sim’}, \text{‘imi’}, \text{‘mil’}, \text{‘ila’}, \text{‘lar’}, \text{‘ari’}, \text{‘rig’}, \text{‘igy’}\}.$$

We have $Jaccard(x, y) = \frac{|x \cap y|}{|x \cup y|} = \frac{6}{8+8-6} = 0.6$.

A predicate involving sets x and y , the Jaccard similarity, and a threshold τ can be equivalently rewritten into a set overlap constraint: $J(x, y) \geq \tau \iff |x \cap y| \geq \frac{\tau}{1+\tau}(|x| + |y|)$.

Further, we can use the *prefix filter* technique [3] to discard set pairs that cannot meet the similarity predicate by examining only a subset of them.

Definition 2 (Prefix Filter). *Consider that the tokens of all sets are sorted based on a total token order. Let $\text{pref}(x, p)$ be the subset of x containing its first p tokens. Then, for any two sets x and y , we have:*

$$|x \cap y| \geq \alpha \implies \text{pref}(x, \alpha + 1) \cap \text{pref}(y, \alpha + 1) \neq \emptyset.$$

The original overlap constraint only needs to be verified on set pairs sharing a prefix token. Pairs that have no prefix token in common can be safely pruned. For the Jaccard similarity and threshold τ , we can identify all candidate matches of a given set x using $\text{pref}(x, \lfloor (1 - \tau) \times |x| \rfloor + 1)$. We denote this prefix simply by $\text{pref}(x)$. Finally, we pick the token frequency ordering as total token order, thereby sorting the sets by increasing token frequencies in the set collection. Thus, we move lower frequency tokens to prefix positions to minimize the number of verifications.

The above concepts apply to single sets. After mapping the string attributes of a record to sets, we end up with multiple sets (or a family of sets). For simplicity, we use interchangeably the term record to refer to the record itself and its representation as a family of sets.

2.2 Problem Definition

Let R be a collection of records. Each record has a set of string attributes $\mathcal{A} = (a_1, \dots, a_j)$. Let $r.a$ denote the string value of attribute $a \in \mathcal{A}$ in a record $r \in R$. Let p be a similarity predicate of the form $\text{sim}(r.a, s.a) \geq \tau$, where sim is a similarity function and $\tau \in [0, 1]$ is a similarity threshold. Let $\phi : R \times R \rightarrow \{\text{true}, \text{false}\}$ be a similarity expression in disjunctive normal form with clauses $C = \{c_1, \dots, c_m\}$, whose literals are similarity predicates and negations are not allowed, i.e.,

$$\phi = \bigvee_{c \in C} \bigwedge_{p \in c} p.$$

Definition 3 (Similarity Join on Multi-Attribute Data). *Let R be a collection of records and ϕ a similarity expression. A similarity join on multi-attribute data returns all pairs of records $(r, s) \in R \times R$, such that $\phi(r, s) = \text{true}$.*

2.3 Apache Spark

Apache Spark is a popular framework for processing of large-scale datasets on shared-nothing architectures. It provides a fault-tolerant abstraction for in-memory data storage and parallel processing on clusters called *resilient distributed datasets* (RDDs). More specifically, an RDD is an immutable, partitioned collection of objects created by coarse-grained deterministic operations

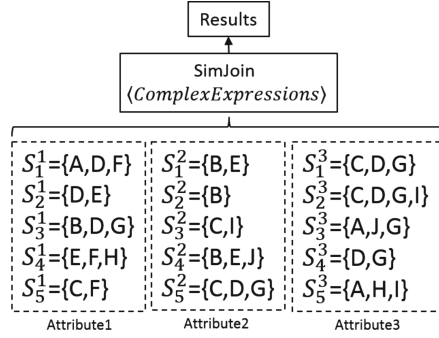


Fig. 1. Set similarity join on multi-attribute data.

called transformations (e.g., map, filter, and groupByKey). Such transformations can be defined either on data in stable storage or other RDDs. Other operations, called actions, return a value to the application (e.g., count) or export data to a storage system (e.g., save). Fault tolerance is provided by logging the transformations that created an RDD into a lineage graph. Spark exposes RDDs through a programming model based on a functional programming API in Scala, Java, Python, or R. In a previous evaluation [15], Spark has been shown to outperform MapReduce [5], another popular distributed data processing framework, in several commonly used workloads.

3 Our Proposed Solution

In this section, we present our solution to the problem of answering similarity joins on multi-attribute data in a distributed environment. We want to support complex similarity expressions over multiple attributes as illustrated in Fig. 1. In the figure, S_n^i represents the set obtained from attribute a_i of record r_n . The main challenge is to incorporate this new similarity join formulation without compromising performance and scalability. We first discuss the general idea of the adopted data partitioning approach. Then, we present our distributed similarity join algorithm. Finally, we give a cost model for selecting the attribute that is used to guide data partitioning.

3.1 Data Partitioning Strategy

We focus on shared-nothing parallel architectures, where multiple (commodity) machines form a cluster connected via a high-speed network. Each machine (or node) is independent, i.e., has its own private memory and disk. Typically, each node may contain multiple multi-core CPUs, to which virtual processors are associated to increase parallelism. We refer to a virtual processor as *worker*.

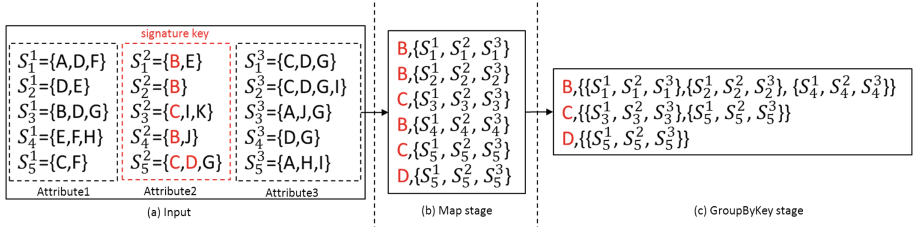


Fig. 2. Data partitioning with signature key.

The data partitioning strategy is crucial in a shared-nothing setting to reduce both communication cost, i.e., the number of records sent across the network, and computation cost, i.e., the number of similarity evaluations performed on each worker. *Signature schemes* have been commonly used as partitioning strategy for similarity joins [6, 17]. Given a similarity function and threshold, set pairs sharing no signature cannot be similar. Thus, an intuitive approach is to use signatures as *partition keys* determining the workers to which a record is sent. Thus, only pairs with a signature in common are sent to the same worker avoiding unnecessary data transmission and similarity evaluations.

Prefix filter can be readily used as a signature scheme, where each prefix token corresponds to a partition key. Because prefix tokens mostly have a low frequency of occurrence, fewer records are transmitted over the network and, in turn, a smaller number of candidate pairs have to be compared in the workers, thereby reducing both communication and computation costs. Note that prefix filter was also used as a signature scheme for distributed similarity joins in reference [17]; however this work does not consider multiple sets.

To use prefix filter in our context, we define the concept of *signature key*. Given a collection of records with an associated schema, the signature key identifies the attribute that will be used for data partitioning, i.e., the prefix tokens derived from this attribute are used as partition keys. Note that the signature key may be involved in more than one predicate in the similarity expression. In such a case, we simply choose the predicate with the highest threshold: higher threshold values imply tighter overlap constraints and, in turn, shorter prefixes. Therefore, a larger number of candidate pairs can be pruned. Figure 2 depicts the use of **Attribute 2** as signature key. Each input record is then associated to each of its prefix tokens as (**key**, **value**) pairs and records associated to the same key are grouped together afterwards. We discuss how to choose the signature key shortly.

3.2 Distributed Similarity Join Algorithm

Our proposed solution is given in Algorithm 1. The algorithm is divided into two main steps: *partition* and *verification*. In the partitioning step (lines 1–2), each input record is passed to the `funcPart` function (lines 6–9), which extracts the prefix tokens from the signature key and returns a list of (**key**, **record**)

Algorithm 1. Distributed similarity join on multi-attribute data.

Input: An RDD R containing a collection of records, a similarity expression ϕ , an integer i identifying the signature key.

Output: An RDD S containing all pairs (r, s) such that $\phi(r, s) = true$.

```

// Partition step
1 list(key, r) ← R.flatMap(funcPart(r, i))
2 list(key, list(r)) ← groupByKey(list(key, r))
// Verification step
3 foreach (key, list(r)) ∈ List(key, list(r)) do
4   S' ← flatMap(evalSim(key, list(r)))
// Collect final result
5 S ← collect(S')
// Functions
6 Function funcPart(r, i)
7   foreach key ∈ pref(r.ai) do
8     list(key, r) ← (key, r)
9   return list(key, r)
10 Function evalSim(key, list(r))
11   foreach candidate pair(r, s) ∈ list(r) do
12     if φ(r, s) = true then
13       S' ← S' ∪ ((r, s))
14   return S'
```

pairs (lines 7–8)—each pair corresponds to a prefix token as partition key and the complete record with all its attributes. Then, the `groupByKey` operation is called to group records with the same key into a list.

In the verification step, record candidates are compared in terms of the similarity expression (lines 3–4). To this end, a list of records sharing the same key is passed to the `evalSim` function (lines 10–14). Similarity evaluation is performed in a distributed and independent manner in each of the *workers* of the cluster. Several filtering techniques can be further employed to reduce the number candidate pairs. Moreover, we can use any main-memory similarity join algorithm, such as `ppjoin` [18] and `mpjoin` [10], to first evaluate the similarity predicate on the signature key attribute; the remaining predicates of the similarity expression are then evaluated on the surviving candidate pairs. Finally, candidate pairs for which the similarity expression evaluates to true are added to the result set and the `collect` action is called to send the result to the Spark application (line 5).

As presented, the algorithm can produce duplicate pairs in the result. Two similar records containing more than one prefix token in common will be sent to different workers and, therefore, appear multiple times in the final result. This problem is avoided by exploiting the global token order. In the verification step, we only compare candidate pairs if the first token in common is equal to the

partition key. Otherwise, if this token is greater than the key, then we are sure this pair has also been sent to at least one other worker and the evaluation can thus be safely interrupted.

3.3 Signature Key Selection

The choice of signature key is a crucial aspect of our approach, as it may significantly impact communication and computation cost. In principle, several heuristics can be used for this purpose. For example, we choose the attribute involved in the predicate with the highest similarity threshold (and hence possibly more selective) or containing shorter strings—these heuristics can also be used to determine the order of evaluating different predicates in the verification step [9]. Another, more sophisticated, alternative is to build performance prediction models based on statistics [16]. However, such approach heavily relies on comprehensive training data, which is difficult to obtain in practice.

We now define a cost model to select the signature key. It is based on frequency statistics of prefix tokens, which implicitly captures the impact of the threshold value and underlying data distribution on the overall performance. In particular, such statistics provide the number of partitions generated (denoted by P), the number of records sent over the network (denoted by X), and the number of similarity evaluations (denoted by Y).

Let T be the a sequence of tokens. The *frequency of moments* of T is defined by $F_k = \sum_{t \in T} df(t)^k$, for each $k \geq 0$, where $df(t)$ is the number of occurrences of token t in T . Assuming that T contains prefix tokens derived for a given signature key, then we have $F_0 = P$, $F_1 = X$, and $F_2 \approx Y$ (the number of similarity evaluations for n records is actually $\binom{n}{2}$). We next define the similarity join cost for a given signature key.

Definition 4 (Similarity Join Cost). *Let c_{comm} and c_{comp} be the cost units of sending a single record over the network and performing a single similarity evaluation, respectively. Given the signature key a , the similarity join cost is*

$$C(a) = X * c_{comm} + Y * c_{comp}.$$

The signature key is defined by the attribute that leads to the lowest cost.

Definition 5 (Signature Key Selection Policy). *Given a set of attributes A , the signature key is given by*

$$a = \arg \min_{a \in A} C(a)$$

We observe that the use of prefix tokens for data partitioning induces important aspects in the distributed computation. The number of partitions is much larger than the number of available workers—the value of P is typically around hundreds of thousands. As a result, several partitions have to be assigned per

worker. Further, data skew is significantly reduced, because large partitions are avoided owing to the token frequency ordering. As a result, the proportion of the overall computation workload regarding any partition is quite low and, thus, a complex load-balancing strategy that assigns multiple workers to a single partition (e.g., [4]) is not crucial. In this context, our cost model assumes an even distribution of the workload among the workers. The design of a load-balancing strategy tailored to our algorithm is left for future work.

4 Experiments

We now present an experimental evaluation of the techniques proposed in this paper. The goals of our study are to evaluate (1) different data partitioning strategies, (2) performance and scalability of our algorithm with varying hardware resources, dataset sizes, and threshold values, and (3) the effectiveness of our cost model for signature key selection.

Table 2. Description of datasets.

Dataset	Type	Records	# Duplicates	Total evaluated
DBLP	Bibliography	250K	5	1.25M
IMDB	Movies	30K	10	300K
DISC	Audio recordings	1M	2	2M

Table 3. Characteristics of the string attributes.

	Attr	Max Len	Avg Len		Attr	Max Len	Avg Len		Attr	Max Len	Avg Len
DBLP	title	340	79	IMDB	title	222	16	DISC	title	676	19
	journal	78	22		actor	51	14		name	623	12
	1st author	41	13		distributor	108	24		1st song	927	17
	2nd author	44	14		director	39	14		2nd song	2689	17
	3rd author	55	14		producer	39	12		3rd song	1236	17

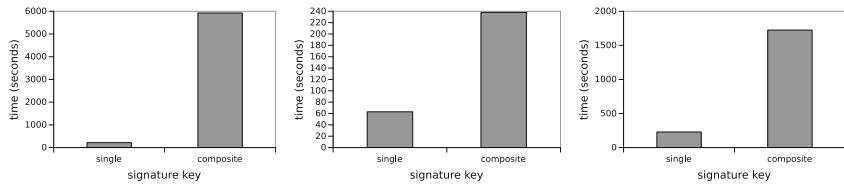
4.1 Experimental Setup

We used three, publicly available, real-world datasets as sources: *DBLP*¹, containing Computer Science publications, *IMDB*², containing movie information, and *DISCOGS*³ (abbreviated to DISC), containing audio recordings.

¹ <http://dblp.uni-trier.de>.

² <http://www.imdb.com>.

³ <http://www.discogs.com>.



(a) DBLP, signature key. (b) IMDB, signature key. (c) DISC, signature key.

Fig. 3. Results of the comparison between different of data partitioning strategies.

In each dataset, we randomly selected records with multiple string attributes. Then, a number of duplicates were generated from each record, i.e., “dirty copies” obtained by performing transformations on string attributes such as characters insertions, deletions or substitutions. Table 2 shows information about the datasets: the initial number of records, the default number of duplicates generated from each record, and the final number of records. In the scalability experiments, we generated larger datasets by increasing the number of records selected from the sources.

Table 3 shows the characteristics of the string attributes used in each dataset, including maximum and average string length. All 5 attributes are used by default. We converted strings to upper-case letters, eliminated repeated white spaces, and generated the corresponding token sets using q -grams of size 3.

A single similarity predicate based on Jaccard was specified for each attribute; the default value of the similarity threshold was 0.85. The resulting similarity expression consists in a single conjunctive clause involving all attributes. We set $c_{comm} = 2$ and $c_{comp} = 1$ in our cost model and used the attribute with the lowest estimated cost as signature key (see Sect. 4.4). Overall performance was measured in average wall-clock time over repeated runs.

We implemented our algorithm using *Oracle Java 8*, *Scala 2.11* and *Spark 2.1.1*. We ran our experiments on 8-node cluster, one node configured as the master and the others configured as slaves. All nodes are equipped with the same hardware resources: *Intel Xeon W3565 Quad-core* processor with 3.2 GHz, 8 MB of CPU cache, 8 GB of main memory, operating system *Ubuntu Server 16.04 LTS*. We used *Hadoop 2.7.3* and YARN configured to use Dynamic Resource Allocation⁴ as cluster manager.

4.2 Comparison of Data Partitioning Strategies

For similarity expressions containing a single conjunctive clause, we can employ an alternative signature scheme for data partitioning. Instead of a single attribute, we can use multiple attributes to define a *composite signature key*.

⁴ <https://spark.apache.org/docs/latest/job-scheduling.html>.

The signature set of a record consists of the combination of the prefix tokens of all attributes that are part of the signature key. On one hand, this approach substantially reduces the computation cost, because two records will belong to the same partition only if they share a token in all attributes. On the other hand, the number of partitions and records transmitted over the network dramatically increases, because the number of signature per record grows exponentially with the number of attributes. Note that this approach was previously adopted in a centralized setting [9] (see further discussion in Sect. 5).

We compared the performance of using single and composite signature keys for data partitioning. For this experiment, we used only the first three attributes of each data dataset. For composite signature keys, we removed partitions containing a single record to reduce communication cost. Figure 3 shows the results of the comparison between single and composite signature keys. Using a single attribute as the signature key was much faster than using composite signature keys on all datasets – up to 12x, 10x, and 4x faster on DBLP, DISC, and IMDB, respectively. Composite signature keys increased lead to an explosion of the number of records transmitted over the network and, thus, the reduction in the number of similarity evaluations did not pay off.

4.3 Varying Hardware Resources, Dataset Sizes, and Thresholds

We now evaluate our algorithm on a varying number of processing nodes, dataset sizes, and threshold values. Figure 4(a)–(c) show performance results with an increasing number of processing nodes. We observe that allocating more processors consistently decreases execution time on all datasets. However, the performance gain decreases as the number of processors increases. This behavior is expected, because the number of processing nodes also increases communication and distributed processing management tends to negatively affect the overall execution time.

Figure 4(d)–(f) plot overall runtime with an increasing number of records. The proposed solution exhibits good scalability: the runtime increases linearly with the number of records on all datasets. Similarity expressions based on more attributes increased runtime on IMDB and DBLP (Fig. 4(g) and (h)), because more similarity predicates have to be evaluated in the verification phase. In contrast, the worst result is obtained for a single attribute on the DISC dataset (Fig. 4(i)). Using a single attribute considerably increased the size of the answer (320 MB for one attribute against around 26 MB when more attributes are considered) and, consequently, increases the overall runtime.

Figures 4(j)–(l) show the results on varying threshold values. As expected, runtime decreases as the threshold increases: greater thresholds lead to shorter prefixes and, thus, less communication and computation cost.

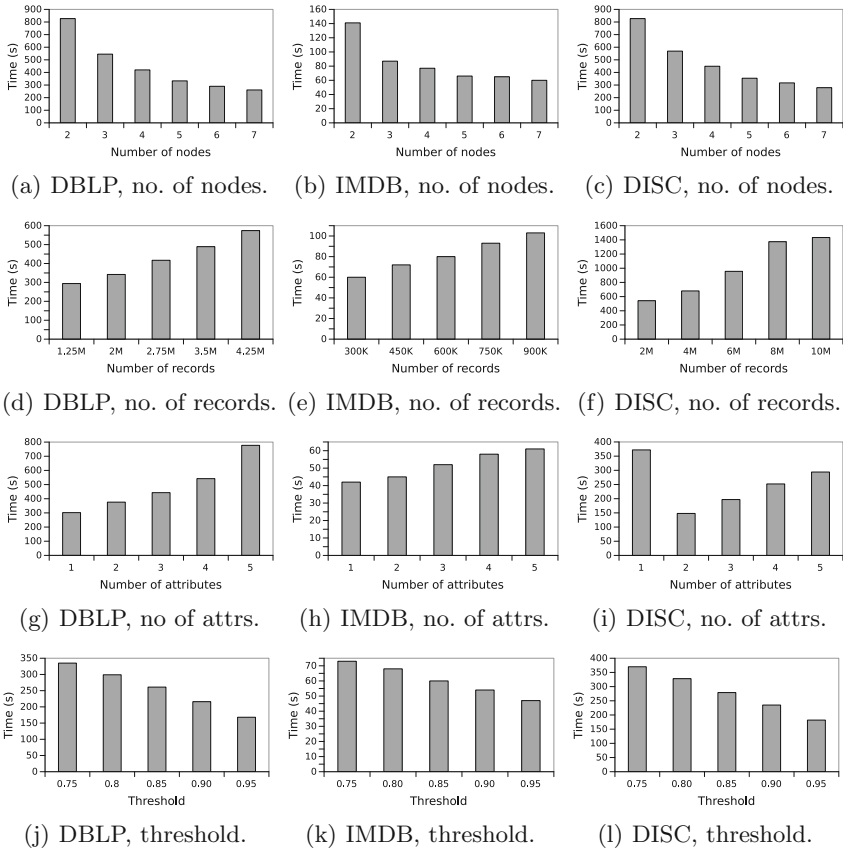


Fig. 4. Results on varying hardware resources, dataset sizes, and threshold values.

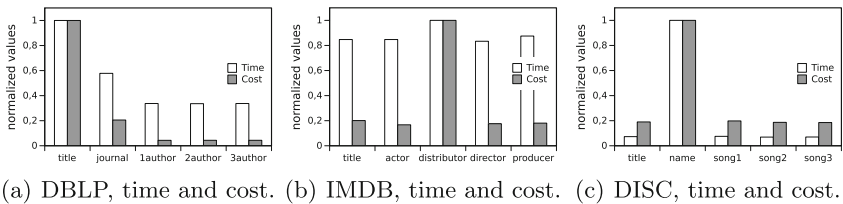


Fig. 5. Signature key selection.

4.4 Evaluation of the Signature Key Selection

Our last experiment evaluates the effectiveness of our cost model in selecting the signature key. Figure 5 shows the normalized values of runtime and estimated cost for different signature keys on all datasets. In all cases, our cost model follows the runtime associated with each attribute, thereby allowing to identify the best signature key for each dataset. Note that higher processing costs can

be associated with attributes containing shorter strings. For example, on the DISC dataset (Fig. 5(c)), the worst overall runtime and higher estimated cost are obtained using `name` as signature key, which contains the shortest strings in average (see Table 3).

5 Related Work

Set similarity joins have been extensively investigated in the literature [1, 3, 10, 14, 18]. A filtering-verification framework supported by inverted lists is prevalently adopted by state-of-the-art algorithms. Most proposals focus in the filtering phase, where various filters are applied to reduce the comparison space, such as prefix filter [3, 14], length filter [14], positional filter [18], and min-prefix [10].

Besides stand-alone algorithms, set similarity joins can be realized using relational database technology. Previous work proposed expressing set similarity joins declaratively in SQL [11] or implementing it as a physical operator within the query engine [3]. Recent work exploits massive parallelism available in modern graphics processing units to speed up similarity join processing [12].

Approximate set similarity joins may miss some valid results to trade accuracy for query time. Locality Sensitive Hashing (LSH) is the most popular technique for approximate set similarity joins [7], which is based on a probabilistic scheme of hashing functions that are approximately similarity-preserving. Min-hash [2] is an example of an efficient LSH scheme for the Jaccard similarity.

Another line of work, which is more closely related to this paper, considers the problem of performing set similarity joins on distributed platforms [6, 13, 17]. Most proposals implemented their algorithms using MapReduce; nevertheless, all these techniques can be implemented in Spark as well. The main ingredient of such approaches, including ours, is the data partition strategy. A common approach is to use a signature scheme, where sets sharing a common signature are sent to the same worker. Like our proposal, reference [17] employs prefix filter to derive signatures. However, only simple similarity predicates over data represented by a single set are considered. The work in [6] introduced a signature scheme based on the size of the symmetric set difference (also known as the Hamming distance). These signature schemes can be directly used in our data partitioning strategy.

A different approach, based on vertical partitioning, was recently presented in [13]. First, the universe of all distinct tokens in the input dataset is partitioned into disjoint segments. The input sets are then split into subsets accordingly and subsets associated with the same segment are sent to the same worker for partial overlap calculation. Partial overlap values between candidate set pairs are then aggregated to obtain the total overlap. This strategy avoids the generation of duplicate sets and produce segments of roughly the same size, thereby ensuring load-balancing among the workers. Unfortunately, it is unclear how to adapt this strategy to our context, which considers multi-attribute data and complex similarity expressions.

To the best of our knowledge, reference [9] is the only previous work that considers set similarity joins on multi-attribute data. Signatures of data objects are generated by the Cartesian product of the prefixes derived from each attribute. A prefix tree index is built to prune candidate pairs based on multiple similarity predicates. Heuristics are presented to reduce the size of the tree index and to determine a predicate order for index construction and similarity expression evaluation in the verification phase. The proposed algorithm is designed for execution on a single machine and only support conjunctions in the similarity expression. In contrast, our algorithm is distributed and supports richer similarity expressions containing conjunctions as well as disjunctions.

6 Conclusions and Future Work

In this paper, we presented a distributed set similarity join algorithm on multi-attribute data, which has received very little attention in the existing literature. Our solution supports complex Boolean expressions based on multiple similarity predicates. We introduced the concept of signature key to enable a simple, but effective data partitioning strategy. Further, we defined a cost model for selecting the signature key associated with the lowest overall runtime. We implemented our proposed approach in Spark and conducted an extensive evaluation on real-world datasets. Experimental results showed that our algorithm is efficient and scalable. As part of future work, we plan to investigate signature schemes tailored to complex similarity expressions.

Acknowledgment. This work was partially supported by Brazilian agency CAPES.

References

1. Bayardo, R.J., Ma, Y., Srikant, R.: Scaling up all pairs similarity search. In: Proceedings of the WWW Conference, pp. 131–140 (2007)
2. Broder, A.Z., Charikar, M., Frieze, A.M., Mitzenmacher, M.: Min-wise independent permutations (extended abstract). In: Proceedings of the STOC Symposium, pp. 327–336 (1998)
3. Chaudhuri, S., Ganti, V., Kaushik, R.: A primitive operator for similarity joins in data cleaning. In: Proceedings of the ICDE Conference, p. 5 (2006)
4. Chu, X., Ilyas, I.F., Koutris, P.: Distributed data deduplication. *Proc. VLDB Endow.* **9**(11), 864–875 (2016)
5. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation, pp. 137–150 (2004)
6. Deng, D., Li, G., Hao, S., Wang, J., Feng, J.: MassJoin: a mapreduce-based method for scalable string similarity joins. In: Proceedings of the ICDE Conference, pp. 340–351 (2014)
7. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the STOC Symposium, pp. 604–613 (1998)

8. Leskovec, J., Rajaraman, A., Ullman, J.D.: Mining of Massive Datasets. Cambridge University Press, Cambridge (2014)
9. Li, G., He, J., Deng, D., Li, J.: Efficient similarity join and search on multi-attribute data. In: Proceedings of the SIGMOD Conference, pp. 1137–1151 (2015)
10. Ribeiro, L.A., Härder, T.: Generalizing prefix filtering to improve set similarity joins. *Inf. Syst.* **36**(1), 62–78 (2011)
11. Ribeiro, L.A., Schneider, N.C., de Souza Inácio, A., Wagner, H.M., von Wangenheim, A.: Bridging database applications and declarative similarity matching. *J. Inf. Data Manag.* **7**(3), 217–232 (2016)
12. Ribeiro-Júnior, S., Quirino, R.D., Ribeiro, L.A., Martins, W.S.: Fast parallel set similarity joins on many-core architectures. *J. Inf. Data Manag.* **8**(3), 255–270 (2017)
13. Rong, C., Lin, C., Silva, Y.N., Wang, J., Lu, W., Du, X.: Fast and scalable distributed set similarity joins for big data analytics. In: Proceedings of the ICDE Conference, pp. 1059–1070 (2017)
14. Sarawagi, S., Kirpal, A.: Efficient set joins on similarity predicates. In: Proceedings of the SIGMOD Conference, pp. 743–754 (2004)
15. Shi, J., et al.: Clash of the titans: mapreduce vs. spark for large scale data analytics. *Proc. VLDB Endow.* **8**(13), 2110–2121 (2015)
16. Sidney, C.F., Mendes, D.S., Ribeiro, L.A., Härder, T.: Performance prediction for set similarity joins. In: Proceedings of the SAC Conference, pp. 967–972 (2015)
17. Vernica, R., Carey, M.J., Li, C.: Efficient parallel set-similarity joins using MapReduce. In: Proceedings of the SIGMOD Conference, pp. 495–506 (2010)
18. Xiao, C., Wang, W., Lin, X., Yu, J.X., Wang, G.: Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.* **36**(3), 15:1–15:41 (2011)
19. Zaharia, M., et al.: Apache Spark: a unified engine for big data processing. *Commun. ACM* **59**(11), 56–65 (2016)

Streaming Data Analysis



Deterministic Model for Distributed Speculative Stream Processing

Igor E. Kuralenok¹(✉), Artem Trofimov^{1,2}, Nikita Marshalkin^{1,2},
and Boris Novikov^{1,2}

¹ JetBrains Research, St. Petersburg, Russia

ikuralenok@gmail.com, trofimov9artem@gmail.com, marnikitta@gmail.com

² Saint Petersburg State University, St. Petersburg, Russia

borisnov@acm.org

Abstract. Users of modern distributed stream processing systems have to choose between non-deterministic computations and high latency due to a need in excessive buffering. We introduce a speculative model based on MapReduce-complete set of operations that allows us to achieve determinism and low-latency. Experiments show that our prototype can outperform existing solutions due to low overhead of optimistic synchronization.

Keywords: Data streams · Distributed processing · Drifting state

1 Introduction

Currently, many applications use stream processing for network monitoring, financial analytics, training machine learning models, etc. State-of-the-art industrial stream processing systems, such as Flink [7], Samza [14], Storm [3], are able to provide low-latency and high-throughput in distributed environment for this kind of problems. However, unlike batch and micro-batch processing, stream processing is inherently non-deterministic [20]. In particular, there is no guarantee that the messages will be processed in the same order and the system produces the same result between any two runs, even if messages are fed to the system with the same monotonically increasing timestamps. Although such behavior is observed in most state-of-the-art stream processing systems, it has several pitfalls:

- It is natural for the user of a software system to expect that two independent runs within the same input data produce exactly the same result. The fact that this contract can be violated is able to cause misleadings and complicates the usage of the system.
- The lack of determinism leads to the loss of reproducibility of the results, that in turn makes testing and verification excessively complicated [19].
- The ability to reproduce predictable results is extremely useful for providing consistency guarantees [16]. The absence of this property forces the usage of heavy transactional protocols to achieve exactly-once semantics [6, 10].

In this work, we introduce FlameStream - stream processing model that is deterministic by design. This property is achieved using strong ordering. The typical way to perform in-order processing is to set up a special buffer in front of each order-sensitive operation [12]. However, extra buffering can lead to latency growth [19], especially if the processing pipeline contains several operations that require ordered input. To avoid this issue, we introduce an optimistic approach for handling out-of-order items that requires single buffer per computational pipeline. Our approach is based on the idea that state can be streamed as an ordinary element. Such approach allows us to generalize speculative computations on the arbitrary MapReduce task. Additionally, it makes the model stateless from the business logic point of view. Our evaluation demonstrates that our method has low overhead and can outperform alternative industrial solution under normal load conditions.

Therefore, the contributions of this paper are the following:

- Definition of a stateful computational model that does not require state handling from the user
- The optimistic schema for deterministic processing and the demonstration of its performance competitiveness.

The rest of the paper is structured as follows: in Sect. 2 we introduce the proposed model and the optimistic approach for handling out-of-order items, the implementation details of the prototype are discussed in Sect. 3 and its performance is demonstrated in Sect. 4, the relevant prior research is mentioned in 5, finally we discuss the results and our plans in Sect. 6.

2 Model

2.1 Motivation

Implementation of deterministic processing is tightly connected to system state management: if user-defined operations are pure functions and the total order is preserved, the processing becomes deterministic. Most of existing stream processing systems have been already supposing that user-defined operations are pure. Instead of providing a handler for external storage they give user operation the state object, provide the user an interface to change the state and finally store the resulting state object after the operation completes [3, 7, 14].

Let B denote a business logic operation, x, Y be input and output items, h , the state handler and s_t , the state object at time t . The change in contract is illustrated in (1). In modern setting B becomes stateless and state management is done on the system side. This change allows the system to implement fault tolerance mechanisms, but it also opens the opportunity to implement deterministic processing.

$$B(x, h) = Y \quad \implies \quad B(x, s_t) = (Y, s_{t+1}) \quad (1)$$

The only difference between a state object s_t and the other items in the stream is that state objects are produced, updated and consumed by the same operation. If a system allows cyclic execution graphs [13] this difference becomes obsolete, as we can transfer state object from operations output to its input. We treat state object as a part of the stream and we call it *drifting state*. Drifting state allows moving fault tolerance logic from user-defined operations to common stream consistency mechanisms.

The second property of the system needed for deterministic processing is total order preservation. This one is quite challenging due to asynchronous nature of the network. On the other hand, we need to care about the order only in the operations that are order-sensitive. All stateless operations are tolerant to the out-of-order items. The more stateless operations we have, the easier the task becomes. Another important note is that calculations are partitioned, and order between items from different partitions does not influence the result. Partition could be calculated at single compute unit, which allows us to implement ordering within single unit instead of system-wide.

Taking into account above considerations we think that modern stream processing problem setup allows us to build a system that is able to provide deterministic processing with low performance overhead. The desired system properties are:

- Computational model should be deterministic by design, i.e., it should produce deterministic results for any pipelines and business logic.
- The performance overhead should be low in comparison with the existing systems.

We will use the following principles for our system:

- Support cyclic execution graph
- Localize state management in terms of system operation type
- A data partition must be processed on a single node
- MapReduce completeness.

2.2 Computational Model

The key concept of FlameStream model is a data stream. It is a sequence of discrete events described by data items, internally represented as $(Payload, Meta)$. The *Payload* is processed by a user-defined code, while *Meta* is handled with FlameStream engine. In particular, the primary purpose of the meta-information is to impose the total order on data items. The *Meta* is assigned at the entry (called *front*) and is discarded at the *barrier* just before the exit.

The stream processing is specified by a logical execution graph. Each node of the graph represents a single operation on data items, and edges describe the routing of data items between operations. Our model allows cycles in the graph while such graphs are commonly assumed to be acyclic (DAGs) [6, 21]. The cycles are required for specification of certain computations (e.g. MapReduce-based)

with our set of operation types (outlined below). Figure 1 shows the example of logical execution graph.

A distributed hardware environment is modeled as a set of *worker* processes. Each worker executes logical execution graph and has an assigned range of hash values used for physical routing of data items to workers. Each operation entry has a user-provided hash function called *balancing function*. This function is applied to the payload of data items and determines partitioning before each operation. After that, the data items are sent to the worker, which is responsible for the associated hash range. Therefore, load balancing explicitly depends on the user-defined balancing functions.

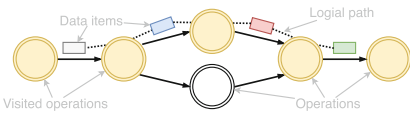


Fig. 1. A logical execution graph

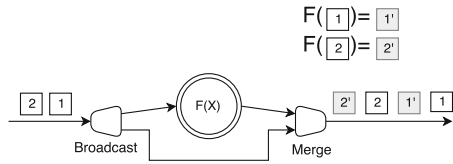


Fig. 2. The ordering model

Data items are totally ordered according to labels assigned to events at the entry as a part of meta-information. All operations preserve this order. Any additional items produced by an operation are placed between the item being processed and the next item. The ordering labels are dropped when items are delivered from the barrier.

The ordering is illustrated in Fig. 2. Data item with payload 1' is the derivative of the item with payload 1, according to operation F . The same is for items with payloads 2' and 2. After merge operation, the order between 1 and 2 is preserved. Furthermore, 1' follows 1, and 2' follows 2.

The list of available operations includes:

Map applies a user-defined function to the payload of an input item and returns a (possibly empty) sequence of data items with transformed payloads.

Broadcast replicates an input item to the specified number of operations.

Merge operation is initialized with the specified number of input nodes. It sends all incoming data to the output.

Grouping constructs a single item containing a set of consecutive items that have the same value of partition function. The maximum number of items that can be grouped is specified as a parameter *WindowSize*.

The output item of the grouping has the same ordering label as the last item in the output group. Groupings of different partitions are independent. Grouping is the only operation that has a state.

The following example illustrates the grouping operation. Let the input stream be a series of integers: 1, 2, 3, ..., and the partition function returns for even numbers and 0 otherwise. If the window is set to 3, the output is

$$(1), (2), (1|3), (2|4), (1|3|5), (2|4|6), (3|5|7), (4|6|8), \dots$$

2.3 Drifting State

An important special case of grouping with $WindowSize = 2$ provides for realization of stateful calculations with drifting state technique manifested in Sect. 2.1. Indeed, consider a map operation that follows the grouping and sends its output to the grouping input. This map operation receives a pair of its previous output considered as the state object, and new incoming item from the source stream. The map operation calculates new state object and sends it back as the grouping input.

As an example, let us demonstrate a generic MapReduce transformation. The map stage of MapReduce can be expressed in terms of our map operation. The generic reduce stage can be presented as

```

for  $mapped \in values$  do
    accumulator := combine (mapped, accumulator);
end for
return accumulator;
    
```

The *accumulator* is an explicit state that should be kept between subsequent iterations. To implement reduce stage we apply the drifting state technique and make the accumulator value a part of the stream. Figure 3 shows a generic graph for the MapReduce transformation. Map and reduce stages are highlighted with a dashed line.

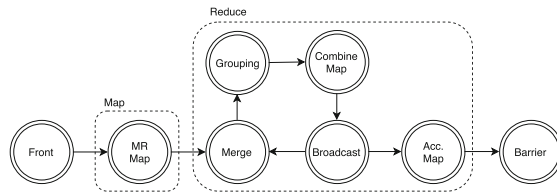


Fig. 3. Logical graph for MapReduce transformations

There are four types of data items in this stream: *input*, *mapped*, *accumulator*, and *reduced*. The operations of the stream have the following purposes:

- The first map operation outputs mapped items according to map stage of MapReduce model.
- The grouping with $WindowSize = 2$ groups the *accumulator* with next *mapped* item.
- The combine map produces a new state of *accumulator* to be sent to grouping.
- The final map converts *accumulator* into final reduce output.

Ordering rules guarantee that each *accumulator* item always arrives at the grouping right before next not yet combined mapped item. The cycle gives the ability for new accumulator items to get back in the grouping operation. Thereby, the stream reacts to each input item by generating new reduced item, which contains the actual value of the reduce stage.

2.4 Deterministic Computations

Deterministic execution is a desired property of any distributed system.

To restrict outputs to only one possible result, we impose the following restrictions on our model:

- We require map function to be pure: return value is only determined by its input values, without observable side effects
- We impose a strict ordering requirement on the grouping input.

While the former can be satisfied by moderating the business logic, the latter is foreign to the distributed systems: it is hard to ensure the right order of delivery due to asynchrony inherent for a distributed system. There are two most common methods that are used to implement order-sensitive operators: in-order processing (IOP) [4,9] and out-of-order processing (OOP) [12]. According to IOP approach, each operation must enforce the total order on output elements. This method does not scale well, because it requires buffering before each, even stateless, operation within pipeline until the total order is reached [12]. OOP is an approach that does not require order maintenance if it is not needed. In the case of ordering requirements, OOP buffers input items until a special condition is satisfied. This condition is based on progress indicators such as punctuations [17], low watermarks [1], or heartbeats [15].

Some state-of-the-art stream processing systems adopt OOP [6], but they suppose that items must be buffered before each order-sensitive operation if deterministic results are required. In our system, we use an optimistic approach for handling out-of-order items, that is based on OOP, but require single buffer per computational pipeline, no matter how many stateful operations it contains.

Only the grouping operation retains a dependency on the order of incoming items. Within the optimistic approach, we accept the fact that grouping can produce incorrect output, but we guarantee that all correct groups are eventually produced. To eventually produce all correct tuples, we use an approach called *repair*. If an item preceding already processed items arrives at the grouping operation, all items starting with the just arrived one are re-processed, and *tombstones* (invalidator) are generated for all items produced earlier for all input items processed before the new one. A tombstone for an item is the same item marked with a flag in its meta-information. Hence, it traverses over the same physical path as the invalidated item.

An example of grouping repair is shown in Fig. 4. The green item is out-of-order. The output consists of the new valid items (1,2) and (2,3) and the tombstone $(1,3)_{tomb}$ for the previously generated item.

The barrier keeps outgoing items on hold and filters out invalid elements, when corresponding tombstones arrive. As soon as no tombstones preceding certain point cannot arrive anymore, items are delivered up to this point. More details can be found in Sect. 3.

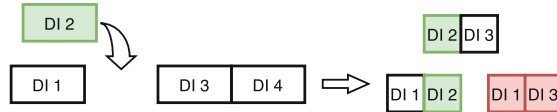


Fig. 4. The repair in grouping with *WindowSize* = 2. (Color figure online)

3 Implementation

FlameStream is implemented in Java, using Akka framework for messaging. There are several main components within the implementation:

Data producers and data consumers are deployed separately and play the role of data source and data sink correspondingly.

Graph is a component that is deployed on each node and executes a computational pipeline defined by a logical graph. Operations within the same node communicate with each other via direct function calls for performance optimization.

Barrier filters out invalid data items. Besides, it delivers output items to data consumers.

Acker tracks data items within the stream. Its functionality is detailed further.

Apache ZooKeeper is used for cluster management. The usage of ZooKeeper mitigates the need for the dedicated master node.

Persistent storage is needed for recovery in case of failures.

The overall scheme of the system components is shown in Fig. 5.

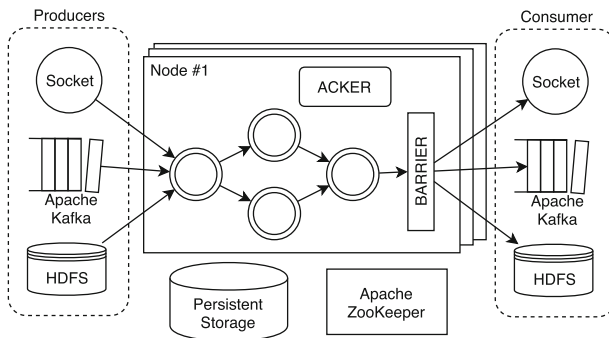


Fig. 5. The overall scheme of the system components

3.1 Ordering Model

The meta-information of data item is implemented as a tuple of a *global time*, a *trace*, *child ids* and a *tombstone flag*.

$$Meta := (GlobalTime, ChildIds[], Trace, IsTombstone)$$

Global time is assigned to data item once the item enters the system. It is a pair of logical time and the identifier of the front. The identifier is used to resolve time collisions within different fronts. It is important to notice that we do not rely on any clock synchronization between nodes. The only implication of the clock skew is the system degradation regarding latency: 1 ms of the fronts clock difference appends 1 ms to minimal latency.

Each map operation can produce multiple items from one. An ordinal number, child id, is stored in the meta information to differentiate them. *ChildIds* is an array of child ids, that corresponds to all visited map operations.

The global time and child ids are enough to identify data item within a stream if all processing is done in-order. In this case, if we compare global time and child ids lexicographically the meta has the desired properties that were defined in Sect. 2.2.

In order to filter out all invalid data at the barrier, there is a need to match tombstones with corresponding invalid items. However, if any grouping repairs happened during processing, multiple items with the same global time and child ids exist in the stream. To differentiate them without direct payload comparison, there is a *Trace* value stored in the meta-information. The trace is a xor of all physical operations' ids (random 64-bit identifier) visited by item so far. Invalid item and the corresponding tombstone go along the same path because they have the same payload and the balancing functions are deterministic. Therefore, item and the corresponding tombstone can be revealed via trace matching.

3.2 Minimal Time Within Stream

To release an item from the barrier we need to ensure that there are no in-flight tombstones for that item, i.e., tombstones which have been already generated but have not arrived at the barrier yet.

Lemma 1. *For any data item D let $\mathcal{G}(D)$ be its global time. If data item D has global time $\mathcal{G}(D) < \mathcal{G}(F)$ for each in-flight element F , then all tombstones for that item had already arrived at the barrier.*

Proof. Let D_{tomb} be a tombstone for D . According to the definition of the tombstone item, $\mathcal{G}(D_{tomb}) = \mathcal{G}(D)$, hence D_{tomb} is not in-flight.

New tombstones for D cannot be generated because items with global time greater than $\mathcal{G}(D)$ cannot trigger repair that affects D . This implies that if the stream does not contain items D' such that $\mathcal{G}(D') \leq \mathcal{G}(D)$, then all tombstones for D had already arrived at the barrier. \square

Therefore, to output an item from the barrier, we should ensure that there are no items in the stream with the global time less than or equal to the global time of this item.

To track the global time of in-flight items we adopt an idea of *acker task* inspired by Apache Storm [3]. Acker tracks data items using a checksum hash, called *XOR*. When the item is sent or received by an operation, its global time and checksum are sent to the acker. This message is called *ack*. Acker groups acks by a global time and xors received checksum hashes. When an item is sent and later received by the next operation, xoring corresponding *XORs* would yield 0.

Acks are overlapped to nullify *XOR* only when an item arrives at the barrier. That is, ack for receive is sent only after both processing and the ack sending for the transformed item are done, as illustrated in Fig. 6. This technique guarantees that the *XOR* for some global time is equal to zero only if there are no in-flight elements with such global time.

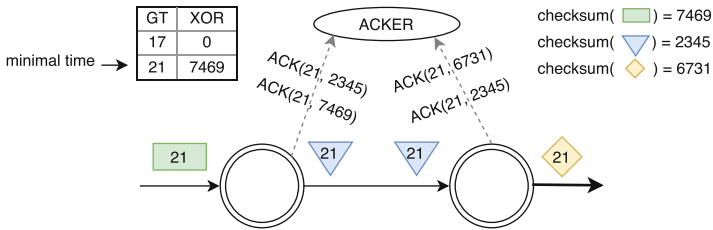


Fig. 6. The example of tracking minimal time using acker

The minimal time within a stream is the minimal global time with non-zero *XOR*. On minimal time changes, acker broadcasts the *new minimal time notification*. Therefore, the barrier can release elements with global time $\mathcal{G}(D)$ once it received a notification with time greater than $\mathcal{G}(D)$.

To ensure that no fronts can generate item with the specific timestamp, each front periodically sends to acker a special message called *heartbeat* indicating that front will not issue items with a timestamp lower than the reported. The value in the ack table can become zero only after the corresponding heartbeat arrives.

4 Experiments

We conducted a series of experiments to estimate the performance of our prototype. We used a problem of building an incremental inverted index as a stream processing benchmark for the following reasons:

- It requires stateful operations
- Computational flow contains network shuffle that can violate the ordering constraints enabling evaluation of our optimistic techniques.
- The workload is unbalanced due to Zipf’s law.

In the real-world, such scenario can be found in freshness-aware systems, e.g., news processing engines.

Building of an inverted index starts with page mapping into the pairs (*word; word positions within the page*), then word positions are reduced by word into the single index update. To avoid index inconsistencies, pairs (*word; word positions within the page*) must be ordered by page id and version before the update of inverted index state. In FlameStream this algorithm is implemented as a conversion of MapReduce transformation outlined in Sect. 2.3. The index update record plays the role of an accumulator.

The latency is defined as a (negated) time difference between the entry of incoming data item and the delivery of all output items generated in response to the incoming one.

Our experiments were performed on the cluster of Amazon EC2 micro instances with 1 GB RAM and 1 core CPU. We used 10000 Wikipedia articles as a dataset.

4.1 Overhead and Scalability

We take the ratio of the total number of items at the barrier to the number of the valid items among them as a key metric for the estimation of the overhead of our prototype and measure it for several system configurations.

The relation between the number of workers, the input document rate, and the ratio is shown in Fig. 8. As expected, the peak of the ratio is achieved when the document per second rate is high, and the number of the nodes is low. This behavior can be explained by the fact that a few workers cannot effectively deal with such intensive load. Nevertheless, the proportion of invalid items reduces if the number of workers grows. The overhead of the optimistic technique under moderate pressure is under 10% for all numbers of workers. These results confirm that the ratio does not increase with the growth of the number of nodes.

The latencies of FlameStream across multiple workers for the fixed document rate of 15 rps are shown in Fig. 7. This experiment demonstrates that latency is stable under the growth in the number of workers. These experiments show that our method is scalable with proper optimization of the system setup.

4.2 Performance Comparison with an Industrial Solution

Apache Flink is chosen for latency comparison in this experiment because it is state-of-the-art stream processing system that provides similar functionality and achieves low latency in the real-world scenarios [8].

For Apache Flink, the algorithm for building the inverted index is adopted by the usage of *FlatMapFunction* for map step and stateful *RichMapFunction* for reduce step and for producing the change records. Order enforcing before reduce is implemented using custom *ProcessFunction* that buffers all input until corresponding low watermark is received. Watermarks are sent after each document. The network buffer timeout is set to 0 to minimize latency.

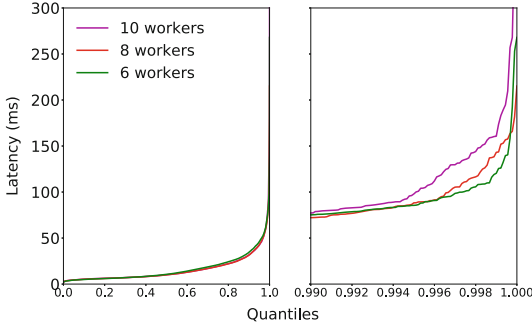


Fig. 7. FlameStream latency distribution (left), high quantiles (right)

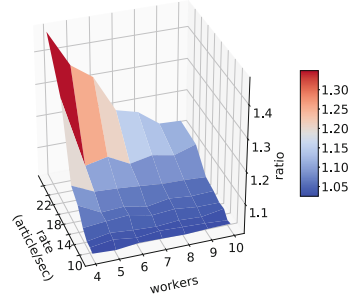


Fig. 8. The relation between the number of workers, the average input rate, and the repair ratio

We compare 50th, 75th, 95th, and 99th percentile of distributions, which clearly represent the performance from the perspective of the users’ experience.

The comparison of latencies between FlameStream and Flink within 10 nodes and distinct document rates is shown in the (a) plot in Fig.9. In this case, FlameStream provides better latency even under high load. These results confirm that optimistic approach for deterministic processing is able to yield lower latency than conservative techniques. The reason for that can be that Flink starts to update index only after the buffer before reduce stage is flushed, while FlameStream flushes its barrier right before data are delivered. Low watermarks go along the stream and can be delayed by long-running operations, while acker processes ack messages independently. It is confirmed by (c) plot in Fig.9, which compares waiting time in Flink buffer and FlameStream barrier.

The (b) plot in Fig.9 compares latencies between FlameStream and Flink within 5 nodes and distinct document rates. Flink outperforms FlameStream under extreme load. Such behavior follows from the fact that FlameStream creates significant overhead under very high load. This result evidently is in line with measurements of the overhead in Fig.8. Nevertheless, FlameStream demonstrates better latency if the load is moderate.

Thus, Flink can be more appropriate if there is a need to optimize computational resources under a fixed load, but the demands on latency are not very strict, or determinism is not required. FlameStream is more relevant for cases when low latency and determinism are strict requirements, but an allocation of additional resources is not a problem.

5 Related Work

Deterministic Processing and Handling Out-of-Order Items: Research works on this topic analyze different methods, but most of them are based on buffering. K-slack technique can be applied, if network delay is predictable [5].

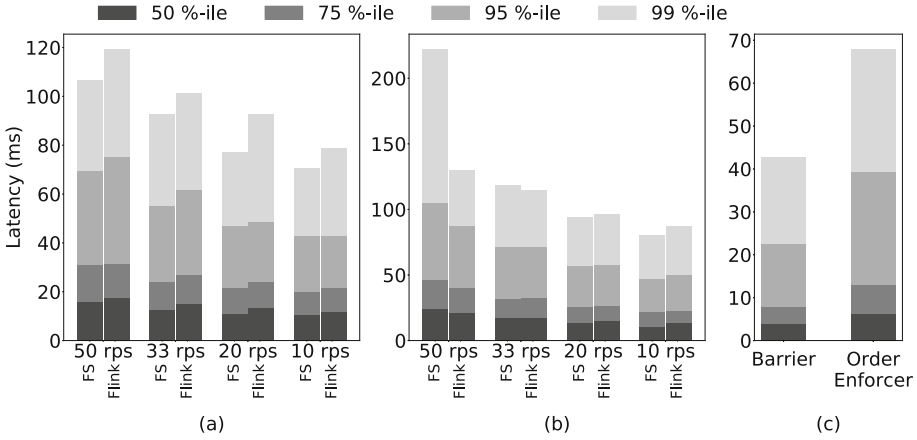


Fig. 9. The comparison in latencies between FlameStream and Flink, (a) - 10 workers, (b) - 5 workers. (c) - the waiting time: Flink order enforcer vs. FlameStream barrier

The key idea of the method is the assumption that an event can be delayed for at most K time units. Such assumption can reduce the size of the buffer. However, in the real-life applications, it is very uncommon to have any reliable predictions about the network delay. IOP and OOP architectures [4, 9, 12] are discussed in Sect. 2.4. The mechanism to control the trade-off between determinism and low latency is proposed in [19]. However, such approach only provides for relaxing determinism properties to achieve low latency if needed.

Optimistic techniques are less covered in the literature. In [18] so-called *aggressive* approach is proposed. They introduced an idea of deletion messages that is very similar to our tombstone items. However, authors describe their idea in an abstract way and do not provide any techniques to apply their method for arbitrary operations. Another optimistic strategy is detailed in [11]. This method is probabilistic: it guarantees the right order with some probability. In addition, it supports only a limited set of query operators.

Data Flow: One specific detail of our computational model is cyclic data flow graphs. Naiad [13] by Microsoft Research provides an implementation of this idea. Nevertheless, Naiad applies cycles only for iterative computations and allows for each operation to have its own state. Another similar concept to Naiad is the usage of logical timestamps to monitor progress. However, to propagate the latest timestamp the pessimistic approach of notifications broadcasting is defined. Therefore, with the assumption of infrequent out-of-order items, our optimistic behavior is more relevant.

In our model, map and group operations are used as core processing primitives. Google Dataflow [2] provides the same idea. The primary distinction is that Google Dataflow has different state model which does not support MapReduce stream processing tasks. Additionally, this model provides different window

types for grouping. FlameStream grouping is aligned with fixed-sized sliding window, but it is possible to implement other kinds of windows by using cycle and grouping with window-affiliation hash.

6 Conclusion and Future Work

We recognized that the lack of determinism implies serious difficulties within the usage of stream processing systems. In this paper, we presented stream processing model that resolves this issue. Particularly, it has the following properties:

- Novel optimistic approach for handling out-of-order items is applied to achieve determinism.
- State management is implemented using drifting state technique, which allows the state to be the part of the stream in the form of an ordinary data item. Such approach relieves the user from the direct state handling.

We implemented the prototype of the proposed model and deeply analyzed its performance and imposed overhead. The series of benchmarks within different computational layouts demonstrated the scalability of the proposed framework. These experiments also showed that our system can outperform the alternative.

In the future, the following features are planned to be implemented:

- Fault tolerance, and, hence, at least once and exactly once guarantees. As it was mentioned above, the property of determinism can simplify the implementation of consistency guarantees, so this is a top-priority task for us.
- Acker can be isolated by hash range. This change allows us releasing from barrier independently also known as early key availability.

References

1. Akidau, T., et al.: Millwheel: fault-tolerant stream processing at internet scale. Proc. VLDB **6**(11), 1033–1044 (2013)
2. Akidau, T., et al.: The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proc. VLDB **8**(12), 1792–1803 (2015)
3. Apache storm, October 2017. <http://storm.apache.org/>
4. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. The VLDB J. **15**(2), 121–142 (2006)
5. Babu, S., Srivastava, U., Widom, J.: Exploiting k-constraints to reduce memory overhead in continuous queries over data streams. ACM Trans. Database Syst. **29**(3), 545–580 (2004). <https://doi.org/10.1145/1016028.1016032>
6. Carbone, P., Ewen, S., Fóra, G., Haridi, S., Richter, S., Tzoumas, K.: State management in apache flink®: consistent stateful distributed stream processing. Proc. VLDB **10**(12), 1718–1729 (2017)
7. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., Tzoumas, K.: Apache flink: stream and batch processing in a single engine. Bull. IEEE Comput. Soc. Tech. Committee Data Eng. **36**(4) (2015)

8. Chintapalli, S., et al.: Benchmarking streaming computation engines: storm, flink and spark streaming. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 1789–1792, May 2016
9. Cranor, C., Johnson, T., Spataschek, O., Shkapenyuk, V.: Gigascope: a stream database for network applications. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, SIGMOD 2003, pp. 647–651. ACM, New York (2003). <http://doi.acm.org/10.1145/872757.872838>
10. Jacques-Silva, G., et al.: Consistent regions: guaranteed tuple processing in IBM streams. *Proc. VLDB Endow.* **9**(13), 1341–1352 (2016)
11. Li, C.W., Gu, Y., Yu, G., Hong, B.: Aggressive complex event processing with confidence over out-of-order streams. *J. Comput. Sci. Technol.* **26**(4), 685–696 (2011). <https://doi.org/10.1007/s11390-011-1168-x>
12. Li, J., Tufte, K., Shkapenyuk, V., Papadimos, V., Johnson, T., Maier, D.: Out-of-order processing: a new architecture for high-performance stream systems. *Proc. VLDB Endow.* **1**(1), 274–288 (2008)
13. Murray, D.G., McSherry, F., Isaacs, R., Isard, M., Barham, P., Abadi, M.: Naiad: a timely dataflow system. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, SOSP 2013, pp. 439–455. ACM, New York (2013). <http://doi.acm.org/10.1145/2517349.2522738>
14. Noghabi, S.A., et al.: Samza: stateful scalable stream processing at LinkedIn. *Proc. VLDB Endow.* **10**(12), 1634–1645 (2017)
15. Srivastava, U., Widom, J.: Flexible time management in data stream systems. In: Proceedings of the PODS, pp. 263–274. ACM, New York (2004)
16. Stonebraker, M., Çetintemel, U., Zdonik, S.: The 8 requirements of real-time stream processing. *SIGMOD Rec.* **34**(4), 42–47 (2005)
17. Tucker, P.A., Maier, D., Sheard, T., Fegaras, L.: Exploiting punctuation semantics in continuous data streams. *IEEE Trans. Knowl. Data Eng.* **15**(3), 555–568 (2003). <https://doi.org/10.1109/TKDE.2003.1198390>
18. Wei, M., Liu, M., Li, M., Golovnya, D., Rundensteiner, E.A., Claypool, K.: Supporting a spectrum of out-of-order event processing technologies: from aggressive to conservative methodologies. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, pp. 1031–1034. ACM, New York (2009)
19. Zacheilas, N., Kalogeraki, V., Nikolakopoulos, Y., Gulisano, V., Papatriantafilou, M., Tsigas, P.: Maximizing determinism in stream processing under latency constraints. In: Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, pp. 112–123. ACM, New York (2017)
20. Zaharia, M., Das, T., Li, H., Shenker, S., Stoica, I.: Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In: Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing, HotCloud 2012, p. 10. USENIX Association, Berkeley (2012)
21. Zaharia, M., et al.: Apache spark: a unified engine for big data processing. *Commun. ACM* **59**(11), 56–65 (2016)



Large-Scale Real-Time News Recommendation Based on Semantic Data Analysis and Users' Implicit and Explicit Behaviors

Hemza Fichel^(✉), Mohamed Ramzi Haddad^(✉), and Hajer Baazaoui Zghal^(✉)

Riadi Laboratory, École Nationale des Sciences de l'Informatique,
University of Manouba, Manouba, Tunisia
hemza.fichel@ensi-uma.tn, haddad.medramzi@gmail.com,
hajer.baazaouizghal@riadi.rnu.tn

Abstract. Online news portals constantly produce a huge amount of content about different events and topics. In such data streams scenarios, delivering relevant recommendations that best suit each user's interests is a challenging task. Indeed, tight-time constraints and highly dynamic conditions in these environments make traditional batch recommendation approaches ineffective. In this paper, we present a scalable news recommendation system that takes into account data semantics, trending topics, users' behaviors and the usage context in order to (1) model news articles, (2) infer users' preferences and (3) provide real-time suggestions. In fact, our proposal is based on the semantic analysis of news articles' content in order to extract relevant keywords and referenced named entities. This information is then used to model users' interests by analyzing their attitudes while interacting with the available content. Moreover, our proposition accounts for the temporal variance of a news article's utility by considering its freshness, popularity and attractiveness. To prove our proposition's quality, scalability and efficiency in real-time data streaming environments, it was evaluated during the CLEF-NEWSREEL challenge connecting recommender systems to an active large-scale news delivery platform. Experiment results show that our system produces high quality and reliable performances in such dynamic environments.

Keywords: News recommendation · Stream-based recommendation
Semantic analysis · Trends · Temporal recommender

1 Introduction

Recommender Systems are *personalized information agents* that help users in finding items matching their preferences [2]. Hereby, an item might be a product to purchase, a movie to watch or a news article to read. In the last years, with the huge growth in the amount of available digital content, considerable attention has been paid to recommender systems.

Every day, millions of news articles are published online. This amount of news makes it difficult to find relevant stories that best suit each individual's interests and preferences. As a consequence, news recommender systems are considered as a potential solution to face this information overload problem. News articles recommender systems have to take into account several domain specific aspects and constraints [9,10,12]. Firstly, the news domain is a highly dynamic area where articles tend to have a short lifecycle which impacts their recommendability and their perceived relevance over the time. Thus, classic recommendation models trained offline on batches of static datasets may not meet the requirements nor be relevant in the long term. Secondly, news articles are continuously created in response to the stream of events and trends. The creation and the consumption rates of news articles may burst in short periods and thus require scalable recommender systems capable of ingestion and processing the high velocity data streams. Thirdly, news portals usually make their content publicly available and do not require users to create accounts or log in before reading news articles. Hence, usage data is not linked to users' profiles or descriptors since they mainly browse content anonymously. This generalizes the cold start problem to all the users base and makes it more challenging to tailor recommendations to their long term interests. Finally, news articles are usually published in an unstructured format which requires natural language processing and semantic analysis to model the available content and users' interests. All these particularities induce the following research challenges for online news recommendation:

1. **Real-time processing and Scalability:** Online news recommenders require robust and low complexity algorithms in order to efficiently deal with the massive amount of news content with respect to the real-time constraint and the available computing power.
2. **Consumption behaviors modeling with incomplete data:** Users behaviors history is an essential knowledge to infer their interests. However, in the news domain, users are mainly anonymous, have no profiles describing them and are not uniquely identified across reading sessions. Thus, usage data mainly reflects a persistent churn and leads to a continuous cold start problem for users. Hence, we believe that news recommender systems should mainly focus on the content, the context and on the fragmented browsing sessions data in order to better model and target users' preferences.
3. **Continuous recommendation and temporal relevance:** As in all recommendation domains, the cold start problem arises also for the newly published news articles. In fact, evaluating the relevance of such items is difficult since they do not have sufficient usage data (e.g. reading or ratings). This problem is more challenging in the news domain due to the continuous creation of news articles in response to current events. Moreover, we believe news novelty is a major factor that drives users' interests. Therefore, the challenge is to propose recommendation approaches capable of rapidly evaluating the relevance of new articles and suggesting them to potential consumers. Any of the existing recommendation approaches that waits until sufficient usage

data is collected to accurately evaluate the relevance of a newly created news article risks exceeding the time frame where it is the most interesting and attractive.

In this paper, we propose a hybrid recommender system that exploits a range of text mining and semantic web technologies to better model news articles' content, infer users' preferences and provide personalized recommendations. The system takes into account content-dependent user preferences and the temporal changes in news articles utility and attractiveness. Moreover, the proposed recommender system was evaluated both online and offline on real-world users and content in order to assess its quality and performances in a large scale, highly interactive and real-time scenarios. The evaluation was conducted during the CLEF-NewsREEL challenge on an active large-scale news delivery platform with real news articles and users' feedback within tight real-time constraints.

The rest of the paper is organized as follows. Section 2 gives an overview of related work. Section 3 outlines the proposed recommendation system and its underlying approaches. Section 4 presents a brief overview of the NewsREEL challenge scenario and discusses both the online and the offline experimentations' results. The last section concludes the paper and gives an outlook on future work.

2 Related Work

Providing relevant recommendations is becoming gradually more difficult as online news portals produce an enormous amount of content every day. To face this information overload problem, several recommendation systems have been proposed. We review the most commonly used ones and discuss their specific strengths and weaknesses.

Content-based approaches provide articles that are similar to those previously viewed by users. They utilize the Vector Space Model to describe content and users' interests using keywords weighted by the term frequency-inverse document frequency (TF-IDF) [11, 13]. These approaches are generic and do not take into account the specific characteristics of the news domain.

Collaborative Filtering algorithms are based on the assumption that users who appreciated similar items in the past tend to have similar interests in the future [7]. Although they are the most commonly used, they suffer from the data sparsity and the cold start problem [14] require a high number of positive interactions. This requirement makes them less suitable for news recommendation scenarios due to the highly dynamic and transient conditions in these environments where user are largely anonymous and content is continuously created and rapidly loosing traction.

Stream-based approaches incorporating trending and temporal factors try to cope with the specific characteristics of the news articles recommendation domain. In fact, this application domain imposes several hard constraints on the recommendation approach to adopt due to large volume and velocity of news streams and users' actions. Besides, recommendations need to be generated on the fly under tight time constraints in order to assist the rapid content

consumption of news articles. Several work in the news recommendation domain focus on extending collaborative filtering approaches with temporal context analysis [3,4]. However, Lommatzsch et al. enumerate several approaches that can incorporate trends and temporal factors in order to improve the quality and the efficiency of news recommendation [10]. Similarly, Garcin et al. suggest that exploiting contextual information helps better model and predict users’ navigational patterns [6].

We believe that currently, there is no universal and generally adopted solution that takes into consideration the dynamics, trends and temporal factors in the news recommendation field. Therefore, in this work, we try to propose and aggregate several specialized models in order to address several recommendation scenarios that need different compromises between personalization and universality, trending and freshness or implicit and explicit users’ behaviors.

3 Proposed System

Figure 1 presents the architecture of the proposed recommender system and its underlying hybrid approach based news articles’ semantics, trends and temporal relevance in order to make suggestions. First, we exploit text mining and semantic web technologies to extract relevant keywords and referenced named entities from each news article’s content as it arrives. The extracted knowledge is then used to model users’ interests by analyzing their attitudes while interacting with the available content. Afterward, the recommender system suggests to users items that are likely to be of their interests based on their preferences models, news articles’ semantics and other influencing factors such as news freshness and trends in order to deal with item-side and user-side cold-start problems.

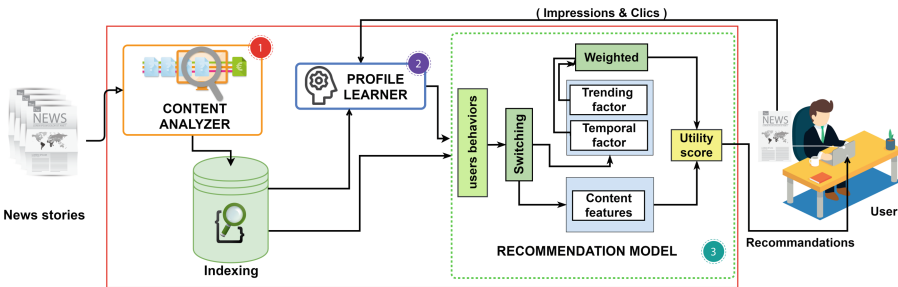


Fig. 1. The architecture of the proposed news recommender system (Some icons in this figure are designed by [Freepik.com](https://www.freepik.com)).

3.1 Content Analyzer

News articles are usually published in an unstructured format which distinguishes them from other items such as movies, books or products. In such circumstances, deriving relevant information from content is a more challenging. In

our proposition, we perform a set of semantic analysis operations on the news articles' content in order to extract the necessary knowledge for users modeling and targeting. Such knowledge is mainly composed of keywords and named entities mentioned in the news story. In order to achieve this objective, we first break news content into basic linguistic terms (i.e. words, numbers, punctuation, etc. . .). Then, these terms are processed by removing non-alphanumeric tokens, lower casing all terms, eliminating all non-informative words (i.e. stop words) and reducing inflectional terms to their root form via lemmatization. This natural language processing phase is useful in order to model articles' content with non-repetitive and relevant terms. In parallel, the content is semantically analyzed in order to understand its meaning and context. Hereby, we try to extract named entities that are mentioned in the text that may refer to events (what happened), times or periods (when), locations (where), individuals or organizations (who). This task is accomplished using the Federated Knowledge Extraction Framework (FOX) [15] which is able to disambiguate the extracted entities against existing Linked Data Knowledge Bases such as DBpedia [8].

In this work, named entities and extracted terms are indexed in a data structure enabling fast queries using the Lucene¹ library. This textual and semantic analysis step for a document d of n words has a linear complexity (i.e. $\mathcal{O}(n)$).

3.2 Profile Learner

In this work, we believe that users' interests may be modeled using the terms and the named entities that are mentioned in the news articles they read. In fact, based on Google Trends statistics for the year 2017², we remark that several top searched terms in the news category refer to named entities. Besides, v et al. suggest that users reading and click patterns in news portals are highly influenced by the stories and the topics that the news articles address [5]. Therefore, we believe that using the news terms and mentioned named entities are the factors that drive users' interests and that they should be used jointly to infer their preferences.

Based on these hypotheses, we model users based on the keywords and the named entities within the articles they interact with. We also suggest that the more frequent a term or a named entity appears in an article, the more importance it should have in the user profile.

In the news domain, since users are mainly anonymous and cannot be precisely tracked across reading sessions, we make no use about the existence of their demographic descriptors and assume that their short-term click history is sufficient to model and predict their interests. Besides, we note that users' clicks do not reflect articles' reading times due to the nature of web navigation where a user may open several news articles in different browser tabs at the same time.

¹ <https://lucene.apache.org/>.

² <https://trends.google.com/trends/yis/2017/GLOBAL>.

3.3 Recommendation Models

The Recommendation model’s objective is to select the most appropriate news articles to recommend to a particular user. This problem may be stated as follows: Given the clicks history of m users ($U = \{u_1, \dots, u_m\}$) over n items ($I = \{i_1, \dots, i_n\}$), and given an active user u with a profile consisting of keywords and named entities extracted from the articles he/she already read, the objective is to predict the most relevant articles that he/she may be interested in reading in the near future.

In this work, we set the following additional challenges that the recommendation approach should respect:

1. The recommendation approach should address the cold start problem on both items and users sides.
2. The recommendation approach should take into account the transient nature of news stories. In fact, new articles have variable lifespans and variable perceived relevance over time.
3. Recommendations should be dynamic and take into account all the available knowledge about the user when they are inferred. This ensures that the approach operates on line (in contrast to batch offline recommendation) and that even the most recent user’s reading are considered for prediction.
4. The recommender system should be able to handle the high velocity of the events generated by news portals (i.e. content creation and users’ clicks).
5. Recommendations should be generated in real-time within a tight response time.

In this work, we propose four recommendation models and a hybridization strategy whose role is to select the most promising depending on the available data. In the next sections, we elaborate on the details of each proposition.

Non-personalized Model (PN_{stat}). In order to tackle the highly sparse users’ data and the cold start problem they face, our approach adopts at first a “universal to everyone” recommendation strategy which converges progressively towards a more personalized strategy while more behavioral data is collected for the user. The non-personalized recommendations are based on news articles recency and popularity which makes it possible to provide potentially interesting suggestions even for new users and articles. The approach analyzes on the fly the data stream by tracking news stories creation and reading events and maintaining an always up-to-date statistics about each stories’ freshness and popularity. This increases the recommendations’ quality by continuously following users’ interests trends and avoiding suggesting outdated stories.

Lommatzsch et al. confirm that the lifecycle of popular news items is very short and that this popularity state remains only for two to three days [10]. Therefore, in order to model popularity (P) and recency (N) as temporal facts that impact the perceived relevance of an article, we propose the PN_{stat} measure, presented in Eq. 1, that utilizes an exponential form to describe the gradual decay of the trending popular news articles as time goes. This measure is then used to

filter the top-N ranking news stories in cold start scenarios where no usage data is available.

$$PN_{stat}(a) = P(a) * e^{-\lambda N(a)} \quad (1)$$

In Eq. 1, for a given article a , $N(a)$ refers to its age while $P(a)$ denotes the number of its readings normalized by the total number of impression. λ represents the trending decay rate allowing a control over the period after which the popularity of the article plummets over time.

The recommendations generated by this model are non-personalized and relatively static overtime. Indeed, once established, the recommendations changes only when the freshly released articles acquire sufficient reads to outperform the most popular items that are aging and losing momentum. Consequently, even if the PN_{stat} model helps avoiding the cold start impact on the recommendations quality, the slow update rate of its suggestions reduces their usefulness and diversity over time.

Personalized Model Based on Implicit Negative Feedback (PN_{dyn}). We extend the non-personalized model PN_{stat} with a feedback mechanism allowing it to adapt its recommendations according to the implicit attitudes that each user could have shown against previous suggestions. Hereby, we take into account the cases where a user, totally aware of a recommended news article, decides to ignore it and not to interact with it. This attitude against that item indicates its low perceived utility for the user and the need for the recommendation approach to acquire this new knowledge implicitly and avoid recommending it again later to that same user. In this context, we consider that the more a story is recommended to a user without inducing a positive attitude (impression or click), the more its utility in time decreases. In order to quantify this feedback, we integrate a new variable $NR(u, a)$ indicating the number of times a user u was recommended the article a . The formalization of this concept would contribute in decreasing the predicted utility of the previously recommended articles as long as are ignored by the user. Consequently, the model would be able to account for this negative and implicit users' feedback and accelerate the replacement of inefficient suggestions that were previously considered as relevant based on their popularity and recency.

Based upon the previous analysis, a dynamic and user-dependent variant named PN_{dyn} is proposed and formulated in Eq. 2. Hereby, δ is a parameter that controls the decay rate of the utility of articles that was recommended several times without inducing a positive attitude.

$$PN_{dyn}(u, a) = P(a) * e^{-\lambda N(a) - \delta NR(u, a)} \quad (2)$$

This formulation makes it possible to select the articles having the best compromise between trending and freshness and novelty to the user. Moreover, it has the merit of being adapted to the context of the real-time recommendation thanks to its constant complexity ($\mathcal{O}(1)$).

Semantically Personalized Models (SEM_{term} and SEM_{ne}). In this work, we propose a semantically personalized recommendation model adopting the same approach as in content based filtering. The assumption here is that news consumers’ behaviors are highly influenced by the articles’ topics and mentioned named entities. Hence, based on the semantic knowledge extracted by the content analysis components, this variant tries to build a user profile describing his/her preferences using the distribution of keywords and named entities in the news articles he/she previously read. The main idea is to recommend to each user, the news articles containing the most important tokens (i.e. keywords for SEM_{term} and named entities for SEM_{ne}) appearing in his/her profile.

In this work, users’ profiles and news documents are encoded in a Vector Space Model [13] associating each token with its TF-IDF reflecting its importance. This representation enables the recommendation model to evaluate a news article’s relevance as the alignment between its indexed content and the active user’s profile using the Cosine similarity measure. The cost of calculating such similarity between a news article d and a user profile u is linearly proportional to the size of their representative vectors ($\mathcal{O}(|d| + |u|)$) which is adapted for online, real-time recommendation scenarios.

A Hybrid Recommendation Variant: The (SEM_{hyb} Recommender). In this work, we also propose a hybrid recommendation approach based on the two proposed personalized recommendation models using a switching hybridization strategy [2]. This proposition is based on the hypothesis that those previous models may not have consistent performance for all types of users. In fact, each model is specialized in different scenarios and utilizes different knowledge sources. Hence, selecting one model as appropriate in a current situation, based on a switching criterion may alleviate the weaknesses of each model when used separately without infringing the real-time constraint. Hereby, the switching criterion is based on the available usage data for the active user since recommendations would be provided by the PN_{dyn} model for a new user, by the SEM_{term} model if the user have read articles with no named entities and by SEM_{ne} as soon as named entities appear in his/her reading history (i.e. in the profile).

4 Experimentations

We evaluated our proposals during the NewsREEL live challenge of the CLEF News REcommendation Evaluation Lab³ [1]. With such online evaluation, we were able to validate our recommendation models under real conditions and ensure that our system is able to deliver high throughput while respecting the imposed 100 ms response delay constraint.

4.1 Methodology

The news NewsREEL challenge, is a campaign-style evaluation lab that offers researchers the opportunity to connect their recommendation systems to an

³ <http://www.clef-newsreel.org>.

active large scale news recommender system. This connection is established via the Open Recommendation Platform (ORP) provided by the news recommendation provider Plista GmbH⁴. The platform is able to provide real-time recommendation service for German news portals. As users visit the partner news publishers, ORP randomly forwards recommendation requests to registered participants as shown in Fig. 2.

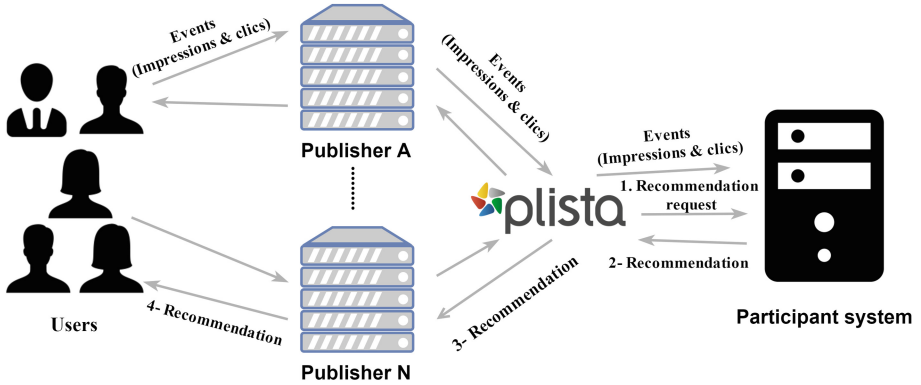


Fig. 2. The figure illustrates the communication between the recommender participant, Plista server and real users.

The recommendations quality of the participating recommender systems is measured by the click-through rate (CTR) referring to the ratio of the generated recommendations that were clicked by users.

NewsREEL also offers an offline task named *NewsREEL Replay* in which participant systems are asked to predict which new articles each user will read in the near future based on a stream of real events issued from historical logs. Hereby, the quality measure, called “offline CTR” is defined as the ratio of successful recommendations that were read by a targeted user in the following 10 min.

4.2 Results and Discussion

In this section, we analyze the evaluation results and discuss the strengths and weaknesses of the implemented models based on their recommendation quality and performances.

Table 1 enumerates the deployed recommender models and their Click-Through Rates for the time period ranging from April 2nd to May 7th 2017.

The results show that both algorithms SEM_{hyb} and PN_{stat} considerably outperformed basic strategies: SEM_{term} and SEM_{ne} . Moreover, we remark that models based on named entities profiles (SEM_{hyb} and SEM_{ne}) provide more

⁴ <https://www.plista.com/>.

Table 1. Click-Through Rates of the proposed recommendation models

Recommender	Recommendation strategy	Profiles type	Requests	Clicks	CTR(%)
SEM_{term}	Cosine similarity between (items, profile)	Keywords extracted from the summaries	7 891	59	0.7477
SEM_{ne}	Cosine similarity between (items, profile)	Named entities extracted from the summaries	34 490	259	0.7509
PN_{stat}	PN ranking ($\lambda = 1$) on the last top-50 recent news	No profile is required	77 723	879	1.1309
SEM_{hyb}	Implement the hybrid recommender strategy	Keywords extracted from the summaries and entities extracted from the full text	75 535	760	1.0062

relevant suggestions than the one based on keywords (SEM_{term}). This can be explained by the fact that users' interests are sometimes driven by the named entities (i.e. personnes, locations, organizations, etc. . .) mentioned in the news.

We should note that the deployed recommenders started providing recommendations from scratch and with no prior data. Over time, our data collection system has processed about 82 336 articles and 3 755 547 users. Moreover, since the evaluation platform only provides news summaries, we had to exploit their corresponding URLs in order to extract named entities from their web pages using two permanent processes extracting named entities newly created articles and old ones. Figure 3 shows the evolution of our SEM_{hyb} model's CTR as more semantic knowledge is extracted by the full text semantic analysis.

The experiment show that the SEM_{hyb} 's recommendation quality depends on the available named entities extracted from the full text of the articles. In other words, users appreciate news citing the same named entities as the ones they have previously read. Moreover, we believe that even if PN_{stat} has better CTR than SEM_{hyb} , the latter may outperform it in the long term. In fact, once users have read all the recent and/or popular news stories, the PN_{stat} model would recommend them all the same older and less popular ones, whereas the proposed SEM_{hyb} would only suggest the ones that are aligned with each ones' personal preferences.

Since it is unpractical fine-tune the models' parameters during the online challenge and the infrastructure needed to deploy multiple configurations at the same time, we extended our evaluation to the offline NewsREEL Replay task. As shown in Figs. 4, 5 and 6, we measure, for all the implemented models, the offline CTR per hour across a three days dataset ranging from February 1st to February 3rd 2016. This dataset contains 18 273 330 recommendation requests belonging to three publishers (*tagesspiegel*, *ksta* and *sport1*) and is characterized by a high cold start percentage around 94%. In this experiment, most recent

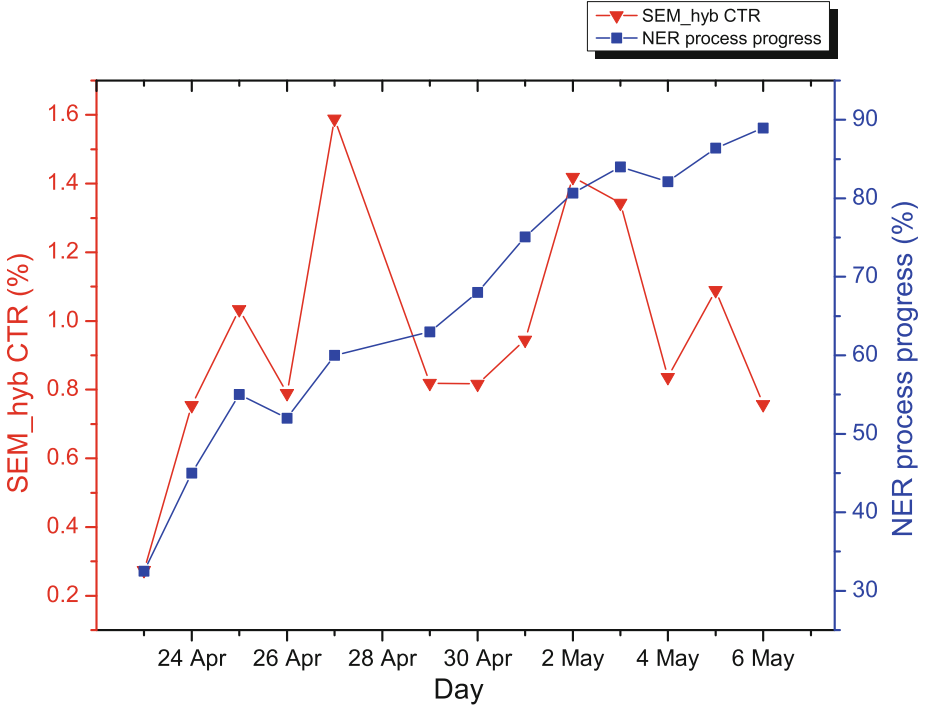


Fig. 3. This figure presents the impact of the availability of more semantic knowledge (right y-axis) on the relevance of the recommendations (left y-axis).

and most popular strategies are used as baseline algorithms. Besides, in the PN_{dyn} recommender, the adopted time unit is the hour and its parameters were empirically estimated (i.e. $\lambda = 1/50$, $\delta = 1/5$).

The results show that baseline recommenders have fluctuating performances for all news portals whereas SEM_{hyb} and PN_{dyn} were able to maintain more stable results over time. SEM_{hyb} shows similar trends as the PN_{dyn} but outperforms it in several times. This is due to the cold start problem and the lack of usage data leading SEM_{hyb} to select PN_{dyn} as its main underlying model. Therefore, the data points where SEM_{hyb} outperforms PN_{dyn} are due to the usage of the semantic sub-models SEM_{term} and SEM_{ne} on the rare identified users having more than one read article. This proves the relevance of textual and semantic analysis in inferring and predicting users' interests.

The experiments show that popularity is the most important explicative factor in general news portals, whereas recency has a fluctuating predicting power (cf. Figs. 5 and 6). We believe that the CTR burst of the most recent strategy is due to the occurrence of a recent event of general interest. In contrast, the recency factor is predominant in the sports news portal while popularity's predictive power is almost null (cf. Fig. 4). We believe this is due to sport news stories being consumed rapidly after their corresponding event and hence rapidly losing

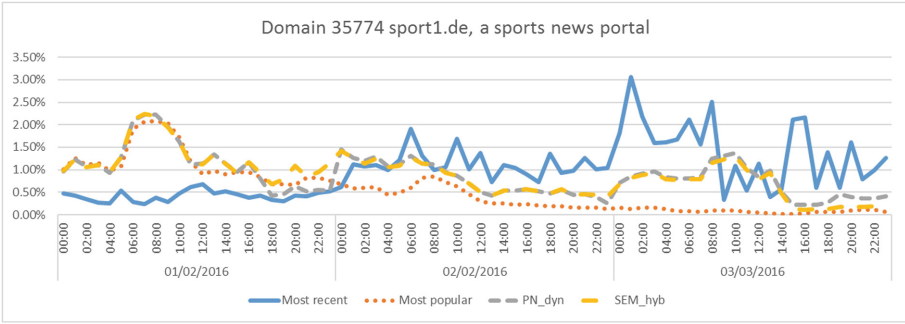


Fig. 4. Offline Click Through Rate by hour of day for publisher sport1

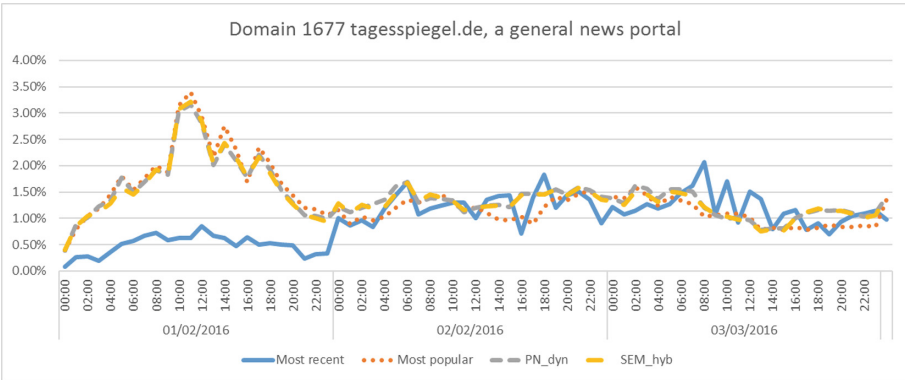


Fig. 5. Offline Click Through Rate by hour of day for publisher tagesspiegel

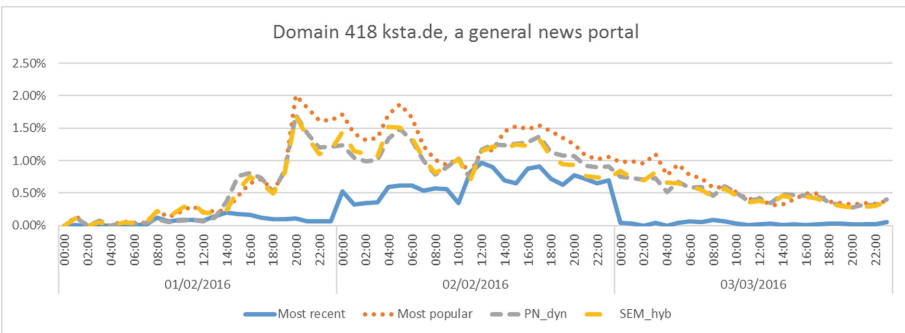


Fig. 6. Offline Click Through Rate by hour of day for publisher ksta

their attractiveness and popularity. This leads us to believe that a deeper domain dependent optimization of λ, δ and the switching conditions of the SEM_{hyb} model may lead to even better results.

In our experiments, all the implemented models were deployed on an ubuntu virtual machine having 4Gb of RAM, 94Gb of storage and running 2 Intel(R)

Xeon(R) CPU E5-26xx processor with 2.0 GHz frequency and 4 Mb of cache. Despite all the semantic analysis and the recommendation workflows, our online hybrid recommender was able to process on average 4 000 requests per minute with an average response time of 47 ms, a mean processor occupancy of 6% and an average memory space of only 240 Mb. In the offline scenario, the system was able to process up to 150 000 events and requests per minute.

5 Conclusions and Future Work

This article enumerates the results and the lessons learned from deploying several recommender systems on a large-scale news recommendation platform. In this work, we studied the usefulness and the effectiveness of several semantic analysis and modeling techniques in a real-world environment with multiple hard constraints. In fact, the news article recommendation domain requires low latency and scalable approaches with limited complexities, minimal response times and a large capability of ingesting the large volumes of data streams that describe the available content and users' interactions. In this context, we proposed a universal solution for news articles recommendation taking into consideration several domain dependent aspect such as content consumption behaviors, dynamics, trends and temporal context. Our proposition also handles content semantics in order to model users' interests using keywords and named entities, hence making the recommendations explainable.

The obtained results show the impact of recency and popularity in recommendations quality and the role of semantic analysis and users explicit and implicit interactions with content in making personalized recommendations. The results are aligned with recent studies' on news recommendation stating that news recommendation approaches based on freshness and popularity are hard to outperform by the ones based solely on items' content [10]. However, we believe that hybridization is the key to overcome the shortcomings of individual strategies.

Our future work will focus on deeper analysis of the correlation between named entities and users' interests. Moreover, we believe that analyzing the contextual dimension of users' behaviors would also enable recommenders better distinguish between long-term and short-term interests and deliver more accurate suggestions.

References

1. Brodt, T., Hopfgartner, F.: Shedding light on a living lab: the CLEF NEWSREEL open recommendation platform. In: Proceedings of the 5th Information Interaction in Context Symposium, pp. 223–226. ACM (2014)
2. Burke, R.: Hybrid recommender systems: survey and experiments. *User Model. User-Adap. Inter.* **12**(4), 331–370 (2002). <https://doi.org/10.1023/A:1021240730564>
3. Campos, P.G., Díez, F., Cantador, I.: Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Model. User-Adap. Inter.* **24**(1–2), 67–119 (2014)

4. Ding, Y., Li, X.: Time weight collaborative filtering. In: Proceedings of the 14th ACM International Conference on Information and Knowledge Management, pp. 485–492. ACM (2005)
5. Esiyok, C., Kille, B., Jain, B.J., Hopfgartner, F., Albayrak, S.: Users' reading habits in online news portals. In: Proceedings of the 5th Information Interaction in Context Symposium, pp. 263–266. ACM (2014)
6. Garcin, F., Dimitrakakis, C., Faltings, B.: Personalized news recommendation with context trees. In: Proceedings of the 7th ACM Conference on Recommender Systems, pp. 105–112. ACM (2013)
7. Koren, Y., Bell, R.: Advances in collaborative filtering. In: Ricci, F., Rokach, L., Shapira, B., Kantor, P.B. (eds.) *Recommender Systems Handbook*, pp. 145–186. Springer, Boston (2011). https://doi.org/10.1007/978-0-387-85820-3_5
8. Lehmann, J., et al.: DBpedia—a large-scale, multilingual knowledge base extracted from Wikipedia. *Semant. Web* **6**(2), 167–195 (2015)
9. Li, L., Wang, D., Li, T., Knox, D., Padmanabhan, B.: SCENE: a scalable two-stage personalized news recommendation system. In: Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 125–134. ACM (2011)
10. Lommatzsch, A., Kille, B., Albayrak, S.: Incorporating context and trends in news recommender systems. In: Proceedings of the International Conference on Web Intelligence, pp. 1062–1068. ACM (2017)
11. Lops, P., de Gemmis, M., Semeraro, G.: Content-based recommender systems: state of the art and trends. In: Ricci, F., Rokach, L., Shapira, B., Kantor, P.B. (eds.) *Recommender Systems Handbook*, pp. 73–105. Springer, Boston, MA (2011). https://doi.org/10.1007/978-0-387-85820-3_3
12. Lv, P., Meng, X., Zhang, Y.: FeRe: exploiting influence of multi-dimensional features resided in news domain for recommendation. *Inf. Process. Manag.* **53**(5), 1215–1241 (2017)
13. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. *Commun. ACM* **18**(11), 613–620 (1975). <https://doi.org/10.1145/361219.361222>
14. Schein, A.I., Popescul, A., Ungar, L.H., Pennock, D.M.: Methods and metrics for cold-start recommendations. In: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 253–260. ACM (2002)
15. Speck, R., Ngonga Ngomo, A.-C.: Ensemble learning for named entity recognition. In: Mika, P., et al. (eds.) *ISWC 2014 Part I. LNCS*, vol. 8796, pp. 519–534. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11964-9_33

Web, XML and Semi-structured Databases



MatBase Constraint Sets Coherence and Minimality Enforcement Algorithms

Christian Mancas^(✉)

Mathematics and Computer Science Department, Ovidius University,
Constanța, Romania
christian.mancas@gmail.com

Abstract. *MatBase* is a prototype data and knowledge base management system based on the Relational, Entity-Relationship, and (Elementary) Mathematical Data Models. The latter distinguishes itself especially by its rich panoply made of 70 constraint types. They provide database and software application designers with the tools necessary for capturing and enforcing all business rules from any sub-universe of discourse, thus guaranteeing database instances plausibility. When dealing with such a wealth of constraint types, both incoherencies and redundancies are possible. As only coherent constraint sets are acceptable and minimal ones are desirable, this paper proposes as fast as possible table-driven algorithms for assisting enforcement of coherence and guaranteeing minimality of such constraint sets, which are implemented in the current *MatBase* versions. The paper also discusses their complexity and utility, both in the study of sets, functions, and relations semi-naïve algebra and first order predicate calculus with equality, as well as, especially, in data modeling, database constraint theory, advanced database management systems, database design, and database software application development.

Keywords: Data modeling · Database constraints theory
Relational constraints · Non-relational constraints
Coherence and minimality of constraint sets · Knowledge base
Integrity checking · Data structures and algorithms for data management
(Elementary) Mathematical Data Model · *MatBase*

1 Introduction

MatBase [10, 11, 13] is a prototype Knowledge and Database Management System (KDBMS) built on top of an existing relational DBMS (RDBMS) and based on both the Relational Data Model (RDM) [1, 4, 10], the Entity-Relationship one (E-RDM) [3, 10, 16], and the (Elementary) Mathematical one ((E)MDM) [6–11, 13], which also embeds Datalog⁻ [1, 11]: its users may define, update, and delete database (db) schemas in any of these three formalisms, which *MatBase* is automatically translating into the other two ones. Relational constraints (i.e. domain/range, not null, (unique) key, referential integrity, and tuple/check) are enforced, in the possible extent, through the host RDBMS [12]. Non-relational ones are enforced through automatically generated trigger-type methods associated to automatically generated forms built on top of corresponding relational fundamental db tables and views.

Just like in any other data model, (E)MDM *constraints*, which are formalizing the business rules that apply in the corresponding sub-universes, are closed first order predicate with equality (FOPC) formulae. Informally, sets of such formulae are said to be *incoherent* if they only allow for empty set instances of at least one set and *coherent* otherwise [5, 10, 11]. Trivially, always empty db instances are of no interest, be it theoretical or practical; consequently, only coherent sets of db constraints are acceptable. When dealing with large constraint sets, even trivial incoherencies are possible by mistake.

Orthogonally, but, as we will see, somewhat dually, coherent constraint sets may be minimal or not. Generally [10–12], a logic formula is *redundant* in a set if it is implied by the rest of the formulae of that set (i.e., in dbs, it holds in any instances in which the rest of the constraints holds); a set containing no redundant formula is called *minimal*. Obviously, especially in dbs, only minimal constraint sets are desirable, as enforcing redundant constraints is not only superfluous, but also time (and memory) consuming.

We have designed and implemented in both current *MatBase* versions (one developed in MS Access and the other in .NET C# and SQL Server) three table-driven algorithms that assist users in enforcing constraint set coherence and guarantee their minimality, which are presented, exemplified, and discussed in this paper.

1.1 A Brief Introduction to (E)MDM

(E)MDM schemes are quadruples made from

- a finite nonempty collection of sets S , partially ordered by inclusion,
- a finite nonempty set of mappings M defined on and taking values from sets of S ,
- a finite nonempty set of constraints C over the elements of S and M , and
- a finite set of Datalog[−] programs P associated to the elements of S and M .

In the context of this paper, only (conventional) db schemas (i.e. triples $\langle S, M, C \rangle$) are of interest (when P is non-empty, the corresponding db is a deterministic deductive one, so a knowledge base).

S is partitioned into the following four blocks: object, value, system, and computed sets.

- *Object* ones are partitioned into *entity* and *relationship*. In relational dbs (rdfs) they are implemented as tables.
- *Value* ones are subsets of (programming/DBMS) data types.
- *System* sets include at least the data types, the empty set, a distinguished countable set *NULLS* of *null values*, and all the sets of the *MatBase* metadata catalog.
- *Computed* sets are obtained from all other types of sets by using semi-naïve sets, functions, and relations algebra operators. In rdfs they are implemented as views.

All mappings in M are defined either on object or on computed sets not based on value ones. In rdfs they are implemented as table/view columns. M is partitioned into the following four blocks: attributes, structural functions (implemented in rdfs as foreign keys), system, and computed mappings.

- *Attributes* are taking values from value sets.
- *Structural functions* from object ones.
- *System* mappings include canonical projections, injections, and unity mappings.
- *Computed* mappings are obtained from all other types of mappings by using semi-naïve sets, functions, and relations algebra operators.

C is partitioned into the following three blocks: set, mapping, and object constraints. Some constraints are associated to corresponding db E-R diagram (E-RD) cycles [8, 11]. The simplest cycles are those of length 1: the *autofunctions* (i.e. of the type $f: A \rightarrow A$). Cycles of length greater than one are made from nodes of three types (*source*, i.e. they are domains of the two mappings that connect them to the cycle, *destination*, i.e. they are co-domains of the two mappings that connect them to the cycle, and *intermediate*, i.e. they are the domain for one mapping and the co-domain for the other one that connect them to the cycle) and may only be of the following three types [8, 11]:

- *commutative* (i.e. having only one source and one destination nodes)
- *circular* (i.e. having only intermediate nodes)
- *general* (i.e. any other cycle than those of types commutative and circular).

The *set constraints* include 16 constraint types partitioned into two blocks: general set and dyadic relation (i.e. binary math relations defined over a set).

The *general set* constraints are sub-partitioned into the following five blocks:

- *inclusion* (e.g. $SEA_HARBORS \subset CITIES$)
- *equality* (e.g. $BIRTH_CERTIFICATES = PEOPLE$)
- *disjointness* (e.g. $MOTHERS \cap FATHERS = \emptyset$)
- *union* (e.g. $UNIVERSITY_PEOPLE = PROFESSORS \cup STUDENTS$)
- *direct sum* (e.g. $PARLIAMENT_MEMBERS = SENATORS \oplus DEPUTIES$).

Dyadic relation ones are sub-partitioned into the following eleven blocks: reflexivity, irreflexivity, symmetry, asymmetry, transitivity, intransitivity, Euclideanity, inEuclideanity, equivalence, acyclicity, and connectivity.

There are 53 *mapping constraint* types that are sub-partitioned into the following five blocks: general mapping, autofunction, homogeneous binary function product (i.e. of the type $f \cdot g: A \rightarrow B^2$), general function product, and function diagram.

General mapping constraints are sub-partitioned into the following five blocks:

- *totality* (i.e. *not null*: $codom(f) \cap NULLS = \emptyset$)
- *nonprimeness* (i.e. it cannot be part of any key; e.g. $Population: CITIES \rightarrow NAT(8)$)
- *one-to-oneness* (*single key*, e.g. $SSN: USCITIZENS \leftrightarrow NAT(9)$)
- *onteness* (e.g. $MountainRange: MOUNTAINS \rightarrow MOUNTAIN_RANGES$)
- *bijection* (e.g. $BirthCertificate: PEOPLE \leftrightarrow BIRTH_CERTIFICATES$).

Autofunction (particular cases of dyadic relations, for which Euclideanity and connectivity do not make sense, as they would violate function definition) constraints are sub-partitioned into the following fifteen blocks:

- *reflexivity*
- *irreflexivity* (e.g. $MedicalDoctor: PEOPLE \rightarrow PEOPLE$)

- *null-reflexivity* (i.e. reflexive for its not null values; e.g. $Capital \circ Country : COUNTRIES \rightarrow COUNTRIES$)
- *null-irreflexivity* (e.g. $Father : PEOPLE \rightarrow PEOPLE$)
- *symmetry*
- *asymmetry*
- *null-symmetry* (e.g. $Spouse : PEOPLE \rightarrow PEOPLE$)
- *null-asymmetry* (e.g. $ReportsTo : EMPLOYEES \rightarrow EMPLOYEES$)
- *idempotency* (e.g. $USCongressRepres : USCITIZENS \rightarrow USCONGRESSMEN$)
- *anti-idempotency*
- *null-idempotency* (e.g. $ReplacementPart : PARTS \rightarrow PARTS$)
- *null-anti-idempotency* (e.g. $Mother, Father$)
- *acyclicity*
- *null-acyclicity* (e.g. $Mother, Father$)
- *canonical surjectivity* (onteness; e.g. $USCongressRepres : USCITIZENS \rightarrow USCONGRESSMEN$).

Homogeneous binary function product ones are sub-partitioned into the following twenty blocks (note that reflexivity does not make sense in this context: why would anybody wish to have two columns of a fundamental table that should store exactly same values?):

- *irreflexivity* (e.g. $Country \bullet NeighborCountry : NEIGHBORS \rightarrow COUNTRIES^2$)
- *null-reflexivity*
- *null-irreflexivity* (e.g. $Husband \bullet Wife : PEOPLE \rightarrow PEOPLE^2$)
- *symmetry* (e.g. $Country \bullet NeighborCountry : NEIGHBORS \rightarrow COUNTRIES^2$)
- *asymmetry*
- *null-symmetry*
- *null-asymmetry* (e.g. $Husband \bullet Wife : PEOPLE \rightarrow PEOPLE^2$)
- *transitivity* (e.g. $Divider \bullet Number : DIVISORS \rightarrow NAT^2$)
- *intransitivity*
- *null-transitivity*
- *null-intransitivity*
- *Euclideanity* (e.g. $Person \bullet BloodRelatedTo : BLOOD_RELATIVES \rightarrow PEOPLE^2$)
- *inEuclideanity*
- *null-Euclideanity*
- *null-inEuclideanity* (e.g. $Husband \bullet Wife : PEOPLE \rightarrow PEOPLE^2$)
- *acyclicity* (e.g. $Prerequisite \bullet Course : PREREQUISITES \rightarrow COURSES^2$)
- *null-acyclicity*
- *connectivity*
- *equivalence* (e.g. $Person \bullet BloodRelatedTo : BLOOD_RELATIVES \rightarrow PEOPLE^2$)
- *null-equivalence*.

Function product ones are sub-partitioned into three blocks:

- *minimal one-to-oneness* (concatenated key)
- *existence* (denoted $f \mid\!-\! g$, i.e. whenever f has a not null value, g must also have a not null value)

- *nonexistence* (dual to existence: for any element of their common domain, only one out of a set of mappings should take a not null value).

Function diagram constraints include the following ten blocks:

- *commutativity (equality)*
- *anti-commutativity*
- *local commutativity* (equivalent to corresponding compound autofunction's reflexivity)
- *local anti-commutativity* (equivalent to corresponding compound autofunction's irreflexivity)
- *local acyclicity* (equivalent to corresponding compound autofunction's acyclicity)
- *local symmetry* (equivalent to corresponding compound autofunction's symmetry)
- *local asymmetry* (equivalent to corresponding compound autofunction's asymmetry)
- *local idempotency* (equivalent to corresponding compound autofunction's idempotency)
- *local anti-idempotency* (equivalent to corresponding compound autofunction's anti-idempotency)
- *generalized commutativity* (particular case of an object constraint only involving mappings of a same function diagram of type general).

Object constraints are FOPC closed Horn clauses (i.e. disjunctions of literals with at most one positive, i.e. unnegated one).

Obviously, for example, set equality is a derived one (from inclusion), just as direct sum (from disjointness and union), equivalence (from reflexivity, symmetry, and transitivity or reflexivity and Euclideanity), totality (from existence), bijectivity (from one-to-oneness and oneness), etc. are. Dyadic relation ones can always be considered as homogeneous binary product ones, where the products are made from their roles (i.e. canonical (Cartesian) projections).

In total, there are only 25 fundamental constraint types in (E)MDM, the remaining 45 being derived. In fact, theoretically, as all constraints are closed FOPC formulas, only the object constraint is actually fundamental. However, as, for example, it is much more easier to assert that *Capital* : *COUNTRIES* → *CITIES* is one-to-one, instead of equivalently asserting that $(\forall x_1, x_2 \in \text{COUNTRIES}) (\text{Capital}(x_1) \neq \text{Capital}(x_2) \Rightarrow x_1 \neq x_2)$, all relevant well-established fundamental math and RDM concepts are considered fundamental in the (E)MDM too.

All five RDBMS provided constraint types are included in (E)MDM too: domain (in co-domain definitions) and referential integrity (from the Key Propagation Principle [2, 10, 12]) implicitly, while not null (totality), keys (minimal one-to-oneness), and tuple/check (extended to object constraints) explicitly.

To conclude about constraints, always discovering and enforcing all existing ones in the sub-universes modeled by dbs is crucial: any existing constraint that is not enforced in a db scheme allows for storing implausible data in its instances. For example, the dyadic relationship *MATCHES* storing results of a double leg championship is connected, irreflexive, symmetric, transitive, and Euclidean; were it storing results for an eliminatory competition, then it is irreflexive, asymmetric, intransitive, and inEuclidean, just like the product *Mother* • *Father*, etc.

- asymmetry \wedge transitivity \Rightarrow acyclicity
- reflexivity \wedge Euclideanity \Rightarrow symmetry \wedge transitivity
- symmetry \wedge Euclideanity \Rightarrow transitivity
- symmetry \wedge inEuclideanity \Rightarrow intransitivity
- symmetry \wedge transitivity \Rightarrow Euclideanity
- symmetry \wedge intransitivity \Rightarrow inEuclideanity
- irreflexivity \wedge transitivity \Rightarrow asymmetry
- symmetry \wedge intransitivity \wedge inEuclideanity \Rightarrow \neg connectivity
- for autofunctions:
 - f reflexive $\Leftrightarrow f = \mathbf{1} \Leftrightarrow f$ total $\wedge f$ symmetric $\wedge f$ idempotent $\Rightarrow f$ bijective
 - f symmetric $\Leftrightarrow f^2 = \mathbf{1}$
 - f canonical surjection $\Leftrightarrow f$ total $\wedge f$ onto $\wedge f$ idempotent
 - f irreflexive $\wedge f$ idempotent $\Rightarrow f$ asymmetric
 - f asymmetric $\vee f$ anti-idempotent $\Rightarrow f$ irreflexive
 - f asymmetric $\wedge f$ idempotent $\Rightarrow f$ acyclic
 - f acyclic $\Rightarrow f$ asymmetric $\wedge f$ irreflexive $\wedge f$ anti-idempotent.

Consequently, not any constraint set is coherent. For example, no autofunction may be both reflexive, symmetric, and anti-idempotent, as the latter two imply irreflexivity. Moreover, not any coherent constraint set is minimal. For example, for any dyadic relation or autofunction the set {acyclic, irreflexive} is not minimal, as irreflexivity is redundant (because acyclicity implies asymmetry, which implies irreflexivity), so it should be replaced with the minimal equivalent set {acyclic}.

2.2 MatBase Knowledge Base Tables Storing Theoretical Results, Redundancies, Coherent, and Incoherent Combinations of (E)MDM Constraint Types

We have added to the *MatBase* metadata catalog a table *THEOREMS* that stores all above (and other) coherence and minimality results. For example, there is a row in it of type “incoherency” having *Description* “symmetric \wedge intransitive \wedge inEuclidean \wedge connected” and one of type “redundancy” having *Description* “symmetric \wedge intransitive \Rightarrow inEuclidean”. Besides *Type* and *Description*, this table also has columns for storing the order in which *MatBase* applies corresponding corollaries, their labels and pages within [11], etc. This table is used for providing users with adequate questions, warnings, and error messages. Its 37 lines (17 with incoherency and 20 with redundancy results) were inserted manually.

For set constraints, only trivial results exist (e.g. two equal sets are each included in the other, hence inclusion is redundant in any set including {equality, inclusion}, two disjoint sets cannot be included one into the other, hence {inclusion, disjointness} is incoherent, etc.). For the rest of the constraint types there are trivial, obvious, and not that obvious results.

We’ve also added to the *MatBase* metadata catalog three tables that store all combinations of constraint types per category – set, mapping, dyadic relations and

homogeneous binary function product, as well as columns for storing whether they are coherent and the first theoretical result that applies. For fast and easy retrieval, these tables have a primary key whose values are computed according to the occurrences of *true* values in the columns storing the corresponding constraint types. The trivially coherent combinations represented by the empty set are not stored.

Other three accompanying tables store corresponding redundancies per coherent combination.

There are only 10 coherent combinations (out of 31) of set constraints stored in the *SCCoherencies* and 5 redundancies in the *SCRedundancies* tables. Abbreviations of the 6 columns of *SCCoherencies* after the primary key x have the following meanings: Ch = Coherent?, S = direct Sum, D = Disjointness, U = Union, E = Equality, I = Inclusion. The unique combination numbers of the primary key x are computed as the decimal equivalents of the corresponding binary ones (where S is multiplied by $2^4 = 16$, U by $2^3 = 8$, ..., and I by $2^0 = 1$).

For example, the {Inclusion}, {Equality}, and {Equality, Inclusion} combinations have 1, 2, and 3, respectively, as x values in *SCCoherencies* and are all coherent, while {Disjointness, Equality, Inclusion} has value 7 ($= 4 + 2 + 1$) for x and is incoherent (as equal sets may not be disjoint, except for the trivial empty set, which is not interesting as only possible table instance). In *SCRedundancies*, for combination having $x = 3$ in *SCCoherencies* there is a row storing the fact that Inclusion is redundant (as, for any two equal sets $S = T$ it is known to the system that $S \subseteq T$ and $S \supseteq T$).

There are 645 rows in the table *HBRCCoherencies* that stores coherent and non-trivially incoherent combinations for the dyadic relation and homogeneous binary function product constraints, out of which 219 coherent ones, and 488 redundancies in the *HBRCRedundancies* tables. Abbreviations of the 12 columns of *HBRCCoherencies* after the primary key x have the following meanings: Ch = Coherent?, IE = InEuclidean, Q = eQuivalence, C = Connected, R = Reflexive, IR = IRreflexive, S = Symmetric, AS = ASymmetric, T = Transitive, IT = InTransitive, E = Euclidean, A = Acyclic. The unique combination numbers x are computed as the decimal equivalents of the corresponding binary ones, just like for all other tables storing constraint type combinations (where IE is multiplied by $2^{10} = 1024$, Q by $2^9 = 512$, ..., and A by $2^0 = 1$).

For example, in *HBRCCoherencies*, combinations {Symmetric} and {Symmetric, Euclidean} have 32 and 34 as values for x (Symmetric being multiplied by 2^5 and Euclidean by 2) and are coherent, while the one for $x = 31$, i.e. {Symmetric, Acyclic} is incoherent (as any acyclic dyadic relation is also asymmetric, so it cannot be symmetric too). In *HBRCRedundancies*, for combination having $x = 42$ in *HBRCCoherencies*, which is {Symmetric, Transitive, Euclidean} (Transitive being multiplied by 2^3), there is a row storing the fact that Euclidean is redundant (as, $Symmetric \wedge Transitive \Rightarrow Euclidean$).

There are 5190 rows in the table *MCCoherencies* that stores non-trivially incoherent combinations for the mapping constraints, out of which 802 coherent ones, and 1437 redundancies in the *MCRedundancies* tables. Abbreviations of the 19 columns of *MCCoherencies* after the primary key x have the following meanings: Ch = Coherent?, C = Connected, Q = eQuivalence, E = Euclidean, IE = InEuclidean, NP = NonPrime,

P = canonical Projection, T = Totality, MI = Minimally Injective (key), O = Onteness, B = Bijection, CS = Canonical Surjection, R = Reflexive, IR = IRreflexive, S = Symmetric, AS = ASymmetric, I = Idempotent (transitive), AI = Anti-Idempotent (intransitive), A = Acyclic. The unique combination numbers x are computed as the decimal equivalents of the corresponding binary ones, just like for all other tables storing constraint type combinations (where C is multiplied by $2^{17} = 131072$, Q by $2^{16} = 65536$, ..., and A by $2^0 = 1$).

For example, in *MCCoherencies*, combinations {Symmetric} and {Symmetric, Anti-Idempotent} have 16 and 18 as values for x (Symmetric being multiplied by 2^4 and Anti-Idempotent by 2) and are coherent, while the one for $x = 15$, i.e. {Symmetric, Acyclic} is incoherent (as any acyclic autofunction is also asymmetric, so it cannot be symmetric too). In *MCRedundancies*, for combination having $x = 13$ in *MCCoherencies*, which is {ASymmetric, Idempotent, Acyclic} (ASymmetric being multiplied by 2^3 and Idempotent by 2), there is a row storing the fact that Acyclic is redundant (as, $ASymmetric \wedge Idempotent \Rightarrow Acyclic$).

All these six tables storing constraint type combinations and their associated redundancies also have a column *Notes* that stores pointers to the *THEOREMS* tables for corresponding incoherency and redundancy reasons, respectively. In the three redundancy tables, the redundant constraint type is stored in a column labeled *Rd* and the constraint combination in column *Combination*.

Trivial incoherencies of type $P \wedge \neg P$ are not stored: for example, combinations 12 to 15 are missing from *HBRCCoherencies*, as they would include both transitivity and intransitivity; similarly, combinations 14 and 15 are missing from *MCCoherencies*, as they would include both idempotency and anti-idempotency. Moreover, due to the supplementary results from [11] proving that absolutely similar results hold for null-type constraints just like for their corresponding well-known basic ones, there is no need to add for the null-type ones corresponding result rows in *THEOREMS* or columns (and corresponding rows) in the three tables storing combinations.

We've added all these seven tables to the MS Access and SQL Server latest versions of the *MatBase* metacatalog, as well as corresponding read-only forms for its normal (i.e. non-system) users. Data for these latter three ones was inserted and updated programmatically, through SQL queries: one such query per table inserted all non-trivially incoherent combinations; then, one query was designed and run for each of the 17 incoherence results, marking corresponding combinations as incoherent; finally, for each of the 20 redundancy results, a query was designed and repeatedly run up until all redundant constraints were inserted too.

By carefully designing the seven tables that are driving these algorithms, only a total of 57 columns and 7,827 rows are stored, although there are $(2^{16} - 1) + (2^{53} - 1) = 65,535 + 9,007,199,254,740,991 = 9,007,199,254,806,526$ possible combinations of the 69 constraint types (i.e. excluding object ones) and 2,830 redundancies.

2.3 *MatBase* ACME, AACE, and AME Algorithms for Coherence and Minimality Enforcement

For each type of non-relational constraint, *MatBase* has a method in its *Constraints* library that enforces the corresponding constraint type for the tables/columns that are its

parameters (together with the current data row and the type of desired operation – add or remove constraint). To enforce a constraint, *MatBase* automatically generates code into the corresponding software db application form classes, for all involved controls and their event types, calling the corresponding library method with the appropriate actual values. These classes are associated with forms; if either such a class or even its host form does not exist, *MatBase* is automatically creating them for the corresponding db table/view. To remove enforcement of a constraint, *MatBase* simply deletes the corresponding calls to the *Constraints* library methods from the corresponding classes.

Figure 1 presents the table-driven pseudocode algorithm *ACME* that is used in *MatBase* to enforce the coherence and minimality of constraint sets.

Obviously, the actual table for which *ACME* computes the values for the primary keys x and from which it reads corresponding data is one of the *SCCoherencies*, *HBRCCoherencies*, or *MCCoherencies*, depending on the type of the parameter constraint c that the user would like to add to or remove from the current constraint set.

The actual algorithm is more complicated and runs in fact each time when users are trying to add a new constraint or to remove or edit an existing one from the db scheme. For keeping things simple, only adding and deleting are included in Fig. 1 (as editing is equivalent to removing the existent constraint and then adding the corresponding modified one).

Editing constraints is managed too, as it may also result in incoherencies: for example, in the *MatBase* geographical demo db there's also a computed set $*WATERS = OCEANS \oplus SEAS \oplus LAKES \oplus RIVERS$, hence, there is also a corresponding direct sum constraint; moreover, $LAKES = *SALTED_LAKES \oplus *SWEET_LAKES$; if a user would like to add $*SWEET_LAKES$ to $*WATERS$, this would cause an inconsistency, as $*SWEET_LAKES$ and $LAKES$ are not disjoint.

ACME rejects both adding, modifying, and removing a constraint that would turn the constraint set incoherent (but also those that cannot be enforced because the current db instance violates it), thus enforcing coherence. Actual *ACME* also automatically adds redundant ones' info (without enforcing them), rejects deleting redundant constraints, automatically deletes redundant ones which are no more implied, and turns fundamental ones into redundant ones whenever this becomes the case (which is also removing their enforcement), thus enforcing minimality too.

For importing legacy dbs, *MatBase* also includes two “bulk” pseudocode table-driven algorithms *AACE*, for assisting enforcement of constraint set coherence, and *AMC*, for enforcing minimality, respectively (see [11] for their pseudocode, as space for this paper is strictly limited to 14 pages). Coherence may not be enforced automatically, as only users should decide what constraints are to be discarded; minimality may, as *MatBase* can automatically remove enforcement of all redundant constraints. They both take as input a set of constraints that might be incoherent and/or not minimal: *AACE* is run first to detect all existing incoherencies (if any) and assist users in removing them; *AMC* runs then to automatically remove any redundancy, if any.

ALGORITHM A1. Constraint Sets Coherence and Minimality Enforcement (*ACME*)

Input: the current (E)MDM scheme constraint set C and a constraint c to be added to or removed from it.

Output: the new coherent and minimal (E)MDM scheme C' .

$C' = C$;

if c is to be removed from C **then**

compute the corresponding x value for the category to which c belongs;

if c is redundant **then** display c & “cannot be removed as it is implied by other constraints, according to “ & $Notes(x)$;

else

$C' = C' - \{c\}$;

remove the calls to the corresponding constraint enforcement method;

compute the corresponding new x value for the category to which c belongs;

remove all redundant constraints that are not implied anymore;

end if;

else

compute the corresponding x value for the category to which c belongs, considering c too;

if $\neg Ch(x)$ **then** display c & “cannot be added as the constraint set would become incoherent, according to “ & $Notes(x)$;

elseif x is missing **then** display c & “cannot be added as the constraint set would become incoherent: it contains \neg “ & c ;

else

if c is not satisfied by the current db instance **then** display c & “cannot be added to the constraint set as the current db instance does not satisfy it!“

else

$C' = C' \cup \{c\}$;

add the calls to the corresponding constraint enforcement method;

mark as true all redundant constraints that are implied by the new C' ;

remove enforcement of any constraint that just became redundant;

end if;

end if;

end if;

End ALGORITHM A1 (*ACME*);

Fig. 1. Algorithm A1 (Constraint Sets Coherence and Minimality Enforcement)

2.4 Examples of Applying the *ACME*, *AACE*, and *AME* Algorithms

Constraints may be added/updated/deleted in *MatBase* through forms for set, relationship, mapping, mapping equalities, etc. For example, in order to add an inclusion, one new line has to be inserted through the *INCLUSIONS* form, by specifying a constraint name (or accepting the system generated one) and choosing the subset and the superset from the corresponding combo-boxes; in order to add acyclicity to a dyadic relation all that is needed is to click on its corresponding check-box in the *RELATIONSHIPS* form; similarly, for mappings there are check-box columns for each

constraint type in the *FUNCTIONS* form; to delete a nonexistence one you just delete its row from the *NON_EXIST_CNSTR* form, etc.

Applying *ACME* on the constraint set of the (E)MDM scheme from 1.2, when trying to add constraint *C15*, it detects the incoherence between it and *C14*: an auto-function cannot be both reflexive and asymmetric, as reflexivity in this context implies symmetry. As, indeed, any root folder belongs to its logic drive, *C14* should be preserved, while *C15* should be rejected. Accepting *C15* too would result in the corresponding OS not being able to store assignation of root folders to logic drives (when formatting them), which, in its turn, would not allow storing any file metadata on that computer (i.e. both *FILES*, *LOGIC_DRIVES*, and *FILE_PROCESSORS* would only allow for empty instances). *AACE* would detect this incoherence too and ask the user which constraint to discard out of *C14* and *C15*.

Moreover, when trying to add constraint *C16*, *ACME* detects that it is redundant: an autofunction that is reflexive is also bijective. As such, *C16* is rejected, as it is implied by *C14*. *AME* would detect this redundancy too and make *C16* redundant.

Consequently, the coherent and minimal subset of the scheme from 1.2 should not include either *C15* or *C16*. As such, this db fragment needs 28 RDM-type constraints (11 domain ones, 7 not null ones, and 10 keys, out of which 4 primary ones, which also embed corresponding not nulls, not counted apart here) and 15 non-relational ones (two nonprimenesses, one acyclicity, one reflexivity, one asymmetry, one idempotency, and all other object constraints).

For example, then, when trying to remove from the db scheme shown in 1.2 the constraint *RootFolder* \circ *LogicDrive* *acyclic*, *ACME* is rejecting it as, according to *C12* and *C13*, this compound mapping is both asymmetric and idempotent, hence acyclic, and redundant constraints cannot be removed from constraint sets. Dually, when trying to add to the same db scheme the constraint *RootFolder* \circ *LogicDrive* *reflexive*, *ACME* is rejecting it too, as asymmetry implies irreflexivity, so accepting it would make the constraint set incoherent.

3 Results and Discussion

3.1 Algorithms Complexity and Optimality

ACME is trivially finishing in finite time as it is constant: it executes at most 8 statements for both removing and adding a constraint, out of which only one reads from and one writes to the *MatBase* metacatalog tables, respectively.

Both *AACE* and *AME* are obviously finishing in finite time too and are linear in the sum of the cardinalities of the object sets collection (i.e. corresponding fundamental db tables/views) and of the mappings set (i.e. corresponding table/view columns).

In fact, for performance reasons (e.g. computing x values for any set and function only once, reading from the corresponding table only once, etc.) *AACE* and *AME* are combined by *MatBase* into a single algorithm that was split here into two only for simplicity of understanding and characterizing them.

Moreover, being that small, table *THEOREMS* fits into only one 4K data block, so it is read only once, when first needed.

Being table-driven, instead of containing huge slow (and very hard to design, develop, and maintain) code for dealing with 9,007,199,254,806,526 cases individually, these three algorithms have a total of only some 80 statements, both in C# and VBA.

3.2 Algorithms Utility

First, analyzing coherence and minimality in (E)MDM constraint sets is interesting even *per se*, within the study of sets, functions, and relations semi-naïve algebra, and first order predicate calculus with equality (FOPC), as it helps getting a better understanding on function diagram commutativity and anti-commutativity, as well as on dyadic relation properties of both single and compound autofunctions and homogeneous binary function products, as well as on the coherent, incoherent, redundant, and minimal closed FOPC sets.

The main utility of these algorithms is, of course, in the realms of data modeling, database constraint theory, advanced database management systems, database design, and database software application development.

ACME detects both incoherencies and redundancies, displays corresponding messages, rejects any attempt to turn the constraint set into an incoherent one, and automatically deletes enforcement of all redundancies.

The algorithm *AACE* for assisting coherence enforcement in legacy dbs is detecting incoherencies, displaying corresponding messages, and only removing those chosen by its users, as it cannot decide which constraints to remove automatically.

The algorithm *AME* for minimality enforcement in legacy dbs detects, displays, and then automatically discards redundancies enforcement.

Therefore, these algorithms are crucial in guaranteeing both the coherence of constraint sets and their minimality.

For example, on a legacy geographical db having 63 object sets, 827 functions, 653 relational and 193 non-relational constraints, *AACE* discovered 10 incoherencies and *AME* 19 redundancies.

4 Conclusion and Further Work

All constraints (which formalize business rules) that are governing the sub-universes modeled by dbs should be enforced in the corresponding dbs' schemas: otherwise, their instances might be implausible. As [2] puts it in its 10th rule (*Data Integrity Is Its Own Reward*) "each 1% data integrity failures will double the amount of time you spend troubleshooting them" and in its 11th one (*The Data Integrity Tipping Point*) "any database which contains 20% or more untrustworthy data is useless and will cost less to replace from source data than to fix". Not only in our opinion, today's businesses cannot be successful if their data is not almost 100% trustworthy.

When dealing with large constraint sets of many types, manually enforcing their coherence (a *sine qua non* condition to allow non-empty db instances) and minimality (an optimization eliminating enforcement of redundant constraints) is very, very hard to do and heavily prone to errors.

Therefore, both *MatBase* current versions also include now the algorithms presented in this paper, for assisting users in enforcing coherence and for automatically eliminating enforcement of redundant (E)MDM constraints, except for the object ones. Moreover, constraint enforcement is done automatically, through code generation, sparing users lot of software developing time and effort.

These algorithms are either constant (the first one), or linear in the sum of the cardinalities of the corresponding db tables, views, and their columns (the latter two), i.e. very fast, table-driven, detecting and dealing with all incoherencies and redundancies for set, dyadic relation, function, and homogeneous function product constraint categories. Were they not being table driven, their complexity would have been exponential in the cardinal of the corresponding constraint set instead.

As object constraints are not considered by the algorithms presented in this paper, their output schemes may still contain both incoherencies and redundancies. For example, if the db scheme from 1.2 were also including a constraint *C20*: $(\forall x \in FILES) (Size(x) = 0 \Rightarrow StartAddress(x) \notin NULLS)$, neither *ACME* nor *AACE* would detect the incoherence between it and constraint *C5*; similarly, if it were also including a constraint *C21*: $(\forall x \in FILE_PROCESSORS)(Ext(Processor(x)) \in \{“com”, “exe”\} \wedge \neg Folder?(Processor(x)))$, neither *ACME* nor *AME* would detect the fact that it is a redundant one, as it is implied by *C19*.

Consequently, further work needs to be done for these algorithms to also consider object constraints. This is not at all an easy task: unfortunately, no pure table-driven solution is possible for them; fortunately, for Horn clauses the implication problem (needed for guaranteeing minimality) is decidable, but it is a NP-complete one if they include, even if only for integers, $x \neq y$ type atoms [15].

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Berkus, J.: Josh's Rules (of Database Contracting) (2007). <http://it.toolbox.com/blogs/database-soup/joshs-rules-of-database-contracting-17253>
3. Chen, P.P.: The entity-relationship model: toward a unified view of data. ACM Trans. Database Syst. **1**(1), 9–36 (1976)
4. Codd, E.F.: A relational model for large shared data banks. Commun. ACM **13**(6), 377–387 (1970)
5. Kueker, D.W.: Mathematical Logic and Theoretical Computer Science. Lecture Notes in Pure and Applied Mathematics Series. CRC Press, Boca Raton (1986)
6. Mancas, C.: A deeper insight into the mathematical data model. In: Proceedings of the 13th ISDBMS International Seminar on DBMS, pp. 122–134. ICI Bucharest, Romania (1990)
7. Mancas, C.: On knowledge representation using an Elementary Mathematical Data Model. In: Proceedings of the 1st IASTED International Conference on Information and Knowledge Sharing (IKS 2002), pp. 206–211. Acta Press, Calgary (2002)
8. Mancas, C.: On modeling closed E-R diagrams using an elementary mathematical data model. In: Proceedings of the 6th ADBIS Conference on Advances in DB and Information Systems, pp. 65–74. Slovak Technology University Press, Bratislava (2002)

9. Mancas, C.: On the equivalence between entity-relationship and functional data modelling. In Proceedings of the 7th IASTED International Conference on Software Engineering and Applications (SEA 2003), pp. 335–340. Acta Press, Calgary (2003)
10. Mancas, C.: Conceptual Data Modeling and Database Design A Completely Algorithmic Approach. Volume I: The Shortest Advisable Path. Apple Academic Press/CRC Press (Taylor & Francis Group), Waretown (2015)
11. Mancas, C.: Conceptual Data Modeling and Database Design: A Completely Algorithmic Approach. Volume II: Refinements for an Expert Path. Apple Academic Press/CRC Press (Taylor & Francis Group), Waretown (2019, in press)
12. Mancas, C., Dorobantu, V.: On enforcing relational constraints in *MatBase*. Lond. J. Res. Comput. Sci. Technol. **17**(1), 39–45 (2017)
13. Mancas, C., Dragomir, S., Crasovschi, L.: On modeling first order predicate calculus using the elementary mathematical data model in *MatBase* DBMS. In: Proceedings of the 21st IASTED International Conference on Applied Informatics (AI 2003), pp. 1197–1202. Acta Press, Calgary (2003)
14. Shoenfield, J.R.: Mathematical Logic. A K Peters/CRC Press (Taylor & Francis Group), Boca Raton/Waretown (2001)
15. Sun, X.H., Kamel, N., Ni, L.: Solving implication problems in database applications. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 185–192. ACM, New York (1989)
16. Thalheim, B.: Fundamentals of Entity-Relationship Modeling. Springer, Berlin (2000)



Integration of Unsound Data in P2P Systems

Luciano Caroprese^(✉) and Ester Zumpano

DIMES, University of Calabria, 87036 Rende, Italy
{l.caroprese,e.zumpano}@dimes.unical.it

Abstract. This paper is placed among the works on semantic peer data management systems. It proposes a different perspective in which a peer of a P2P system may consider its own data to be *more trustable* with respect to data provided by external peers (*sound peer*) or *less trustable* (*unsound peer*). The paper generalizes previous works of the same authors and proposes a new semantics capturing this simple idea.

1 Introduction

The possibility for the users for sharing knowledge from a large number of informative sources, have enabled the development of new methods for data integration [27] easily usable for processing distributed and autonomous data. Due to this, there have been several proposals which consider the integration of information and the computation of queries in an open ended network of distributed peers [3, 6, 8, 9, 21] as well as the problem of schema mediation [25, 26, 28], query answering and query optimization in P2P environments [1, 7, 19, 29]. Many of the approaches proposed in the literature investigate the data integration problem in a P2P system by considering each peer as initially consistent, therefore the introduction of inconsistency is just relied to the operation of importing data from other peers. These approaches assume, following the basic classical idea of data integration, that for each peer it is preferable to import as much knowledge as possible. This basic setting is the one we used in many of our previous works [11, 13–15, 17] in which a different interpretation of mapping rules, that allows importing from other peers only tuples not violating integrity constraints, has been proposed. This interpretation of mapping rules has led to the proposal of a semantics for a P2P system defined in terms of *Preferred Weak Models*. Under this semantics only facts not making the local databases inconsistent can be imported, and the preferred weak models are the consistent scenarios in which peers import maximal sets of facts not violating the integrity constraints. Therefore, the preferred weak model semantics follows the classical strategy of importing as much knowledge as possible, but limiting this to the maximal subset that do not generate inconsistencies.

In this paper, we propose a different perspective and generalize the definition of P2P system in order to allow the possibility for each peer to declare either that its local knowledge is preferred with respect to the knowledge that can

be imported from other peers (*sound peer*) [11, 13–15, 17], or that it gives less preference to its local knowledge with respect to the knowledge that can be extracted from the rest of the system (*unsound peer*). An unsound P2P system is a P2P system containing at least one unsound peer. Note that, an unsound peer is not necessary an inconsistent peer. Basically, it is a peer that joins the P2P system, aims at maximizing its knowledge by importing data from the rest of the system, while maintaining or achieving a consistent state [2, 24] and for some reason has less esteem of its own data with respect to the data provided by others. The semantics of an unsound P2P system is captured by a modified version of the weak model semantics of a correspondent standard P2P system obtained by splitting each unsound peer into two peers. Therefore a generic peer of the system can be:

- *Sound*. In this setting it assumes that its own knowledge is preferable with respect to the knowledge that can be imported from other peers.
- *Unsound*. In this setting it assumes that the knowledge that can be imported from other peers is preferable with respect to the knowledge that can be imported from other peers.

We stress that in all the above cases the final aim of a generic peer, independently of its initial state and of its self esteem, is that of using the P2P environment to maximize as much as possible its knowledge, while maintaining or achieving a consistent state.

Let’s now introduce an example, that will be used as a running example in the rest of the paper.

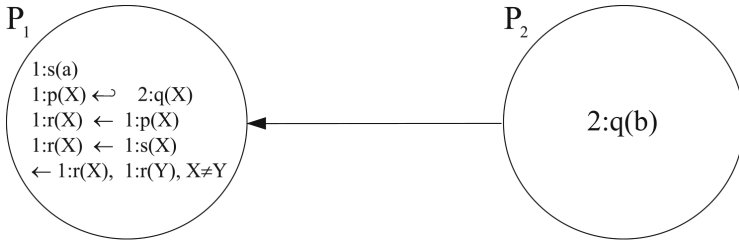


Fig. 1. A P2P system (Example 1)

Example 1. Consider the P2P system \mathcal{PS} depicted in Fig. 1. \mathcal{P}_2 contains the fact 2:q(b), whereas \mathcal{P}_1 contains the fact 1:s(a), the mapping rule 1:p(X) \leftrightarrow 2:q(X), the constraint \leftarrow 1:r(X), 1:r(Y), X \neq Y and the standard rules 1:r(X) \leftarrow 1:p(X) and 1:r(X) \leftarrow 1:s(X).

- if \mathcal{P}_1 is sound, it considers its own knowledge more trustable than the knowledge that can be imported from \mathcal{P}_2 . Then the fact 1:p(b) cannot be imported in \mathcal{P}_1 , as it indirectly violates its integrity constraint. More specifically, 1:p(b) cannot be imported in \mathcal{P}_1 due to the presence of the local fact 1:s(a).

- if \mathcal{P}_1 is *unsound*, it considers its own knowledge less trustable than the knowledge imported from \mathcal{P}_2 . Then the fact $2:q(b)$ is imported from \mathcal{P}_2 and $1:s(a)$ is removed from \mathcal{P}_1 .

Previous example outlines the direction of this paper. It extends the preferred weak model semantics proposed in [11, 13–15, 17] whose basic assumption is that each peer is initially consistent and gives preference to its local data with respect to data imported from the neighborhood and depicts a more general setting in which a generic peer can drive the integration process based on the esteem with respect to its own data.

Organization. Preliminaries are reported in Sect. 2. Section 3 introduces the syntax used for modeling a P2P system and reviews the *Preferred Weak Model* semantics, proposed in [11, 13]. Section 4 proposes a generalization of a P2P system so that each peer can be declared *sound* or *unsound*. Section 5 provides results on the computational complexity of computing preferred weak models and answers to queries. Related work is discussed in Sect. 6. Conclusions and directions for further research are drawn in Sect. 7.

2 Background

We assume that there are finite sets of *predicates*, constants and variables. A *term* is either a constant or a variable. An *atom* is of the form $p(t_1, \dots, t_n)$ where p is a predicate and t_1, \dots, t_n are terms. A *literal* is either an atom A or its negation $\text{not } A$. As in this work we use the *Closed World Assumption*, we adopt *Negation as Failure*. A *rule* is of the form:

- $H \leftarrow \mathcal{B}$, where H is an atom and \mathcal{B} is a conjunction of literals or
- $\leftarrow \mathcal{B}$, where \mathcal{B} is a conjunction of literals.

H is called *head* of the rule and \mathcal{B} is called *body* of the rule. A rule of the form $\leftarrow \mathcal{B}$ is also called *constraint*. A program \mathcal{P} is a finite set of rules. \mathcal{P} is said to be positive if it is negation free. The definition of a predicate p consists of all rules having p in the head.

An *exclusive disjunctive rule* is the form $A \oplus A' \leftarrow \mathcal{B}$ and it is a notational shorthand for $A \leftarrow \mathcal{B} \wedge \text{not } A'$, $A' \leftarrow \mathcal{B} \wedge \text{not } A$ and $\leftarrow A \wedge A'$ ¹. Its intuitive meaning is that if \mathcal{B} is *true* then exactly one between A or A' must be *true*.

It is assumed that programs are *safe*, i.e. variables appearing in the head or in negated body literals are range restricted as they appear in some positive body literal.

An atom (resp. literal, rule, program) is *ground* if no variable occurs in it. A ground atom is also called *fact*. The ground instance of an atom a (resp. literal l , rule r , program \mathcal{P}), denoted by $\text{ground}(a)$ (resp. $\text{ground}(l)$, $\text{ground}(r)$, $\text{ground}(\mathcal{P})$) is built by replacing variables with constants in all

¹ We use for the operator *and* both ‘ \wedge ’ and ‘ \wedge ’.

possible ways. An interpretation is a set of ground atoms. The truth value of ground atoms, literals and rules with respect to an interpretation M is as follows: $val_M(A) = (A \in M)$, $val_M(not\ A) = not\ val_M(A)$, $val_M(L_1, \dots, L_n) = \min\{val_M(L_1), \dots, val_M(L_n)\}$ and $val_M(A \leftarrow L_1, \dots, L_n) = val_M(A) \geq val_M(L_1, \dots, L_n)$, where A is an atom, L_1, \dots, L_n are literals and $true > false$. An interpretation M is a model for a program \mathcal{P} , if all rules in $ground(\mathcal{P})$ are *true* w.r.t. M . A model M is said to be minimal if there is no model N such that $N \subset M$. We denote the set of minimal models of a program \mathcal{P} with $\mathcal{MM}(\mathcal{P})$. Given an interpretation M and a predicate g , $M[g]$ denotes the set of g -tuples in M . The semantics of a positive program \mathcal{P} is given by its unique minimal model which can be computed by applying the *immediate consequence operator* $\mathbf{T}_{\mathcal{P}}$ until the fixpoint is reached ($\mathbf{T}_{\mathcal{P}}^{\infty}(\emptyset)$). The semantics of a program with negation \mathcal{P} is given by the set of its stable models, denoted as $\mathcal{SM}(\mathcal{P})$. An interpretation M is a *stable model* (or *answer set*) of \mathcal{P} if M is the unique minimal model of the positive program \mathcal{P}^M , where \mathcal{P}^M is obtained from $ground(\mathcal{P})$ by (i) removing all rules r such that there exists a negative literal $not\ A$ in the body of r and A is in M and (ii) removing all negative literals from the remaining rules [23]. It is well known that stable models are minimal models (i.e. $\mathcal{SM}(\mathcal{P}) \subseteq \mathcal{MM}(\mathcal{P})$) and that for negation free programs, minimal and stable model semantics coincide (i.e. $\mathcal{SM}(\mathcal{P}) = \mathcal{MM}(\mathcal{P})$).

3 P2P Systems: Syntax and Semantics

This section introduces the syntax used for modeling a P2P system and reviews the *Preferred Weak Model* semantics, proposed in [11,13], in which a special interpretation of mapping rules is introduced.

3.1 Syntax

A peer identifier is a number $i \in \mathbb{N}^+$. A (*peer*) *predicate* is a pair $i:p$, where i is a *peer identifier* and p is a predicate². A (*peer*) *atom* A is of the form $i:p(X)$, where i is a *peer identifier*, $p(X)$ is an atom and X is a list of terms. A (*peer*) *literal* is a peer atom A or its negation $not\ A$. A conjunction $\mathcal{B} = i:p_1(X_1), \dots, i:p_m(X_m), not\ i:p_{m+1}(X_{m+1}), \dots, not\ i:p_n(X_n), \phi$, where ϕ is a conjunction of built-in atoms³, will be also denoted as $i:(p_1(X_1), \dots, p_m(X_m), not\ p_{m+1}(X_{m+1}), \dots, not\ p_n(X_n), \phi)$.

Definition 1. [PEER RULE] A (*Peer*) *rule* can be of one of the following four types:

² Whenever the reference to a *peer predicate* (resp. *peer atom*, *peer literal*, *peer fact*, *peer rule*, *peer standard rule*, *peer integrity constraint*, *peer mapping rule*) is clear from the context, the term *peer* can be omitted.

³ A *built-in atom* is of the form $\theta(X, Y)$, where X and Y are terms and $\theta \in \{<, >, \leq, \geq, =, \neq\}$. It is also denoted as $X\ \theta\ Y$.

1. *(Peer) Standard Rule.*

It is of the form $H \leftarrow \mathcal{B}$, where H is an atom and \mathcal{B} is a conjunction of literals and built-in atoms.

2. *(Peer) Integrity Constraint.*

It is of the form $\leftarrow \mathcal{B}$, where \mathcal{B} is a conjunction of literals and built-in atoms.

3. *Mapping Rule.*

It is of the form $H \leftrightarrow \mathcal{B}$, where H is an atom, \mathcal{B} is a conjunction of atoms and built-in atoms and $i \neq j$. \square

In previous definition, i (resp. j) is the *peer identifier* (resp. *source peer identifier*) of the rule, H is the *head* of the rule and \mathcal{B} is the *body* of rule. The concepts of *ground rule* and *fact* are similar to those reported in Sect. 2. The definition of a predicate $i:p$ consists of the set of standard rules in whose head $i:p$ occurs. A predicate can be of three different kinds: *base predicate*, *derived predicate* and *mapping predicate*. A base predicate is defined by a set of ground facts; a derived predicate is defined by a set of standard rules and a mapping predicate is defined by a set of mapping rules. An atom $i:p(X)$ is a *base atom* (resp. *derived atom*, *mapping atom*) if $i:p$ is a base predicate (resp. standard predicate, mapping predicate). Given an interpretation M , $M[\mathcal{D}]$ (resp. $M[\mathcal{LP}]$, $M[\mathcal{MP}]$) denotes the subset of base atoms (resp. derived atoms, mapping atoms) in M .

Definition 2. [P2P SYSTEM] A *peer* \mathcal{P}_i , with a peer identifier i , is a tuple $\langle \mathcal{D}_i, \mathcal{LP}_i, \mathcal{MP}_i, \mathcal{IC}_i \rangle$, where

- \mathcal{D}_i is a set of facts whose peer identifier is i (*local database*);
- \mathcal{LP}_i is a set of standard rules whose peer identifier is i ;
- \mathcal{MP}_i is a set of mapping rules whose peer identifier is i and
- \mathcal{IC}_i is a set of constraints over predicates defined by \mathcal{D}_i , \mathcal{LP}_i and \mathcal{MP}_i whose peer identifier is i .

A *P2P system* \mathcal{PS} is a set of peers $\{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ s.t. for each source peer identifier j occurring in its mapping rules, $j \in [1..n]$. \square

Given a P2P system $\mathcal{PS} = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, where $\mathcal{P}_i = \langle \mathcal{D}_i, \mathcal{LP}_i, \mathcal{MP}_i, \mathcal{IC}_i \rangle$ for $i \in [1..n]$, \mathcal{D} , \mathcal{LP} , \mathcal{MP} and \mathcal{IC} denote, respectively, the global sets of ground facts, standard rules, mapping rules and integrity constraints, i.e. $\mathcal{D} = \bigcup_{i \in [1..n]} \mathcal{D}_i$, $\mathcal{LP} = \bigcup_{i \in [1..n]} \mathcal{LP}_i$, $\mathcal{MP} = \bigcup_{i \in [1..n]} \mathcal{MP}_i$ and $\mathcal{IC} = \bigcup_{i \in [1..n]} \mathcal{IC}_i$. In the rest of this paper, with a little abuse of notation, \mathcal{PS} will be also denoted both with the tuple $\langle \mathcal{D}, \mathcal{LP}, \mathcal{MP}, \mathcal{IC} \rangle$ and the set $\mathcal{D} \cup \mathcal{LP} \cup \mathcal{MP} \cup \mathcal{IC}$.

3.2 Semantics

This section reviews the *Preferred Weak Model* semantics for P2P systems [11, 13] which is based on a special interpretation of mapping rules.

Given a P2P system $\mathcal{PS} = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, for each peer $\mathcal{P}_i = \langle \mathcal{D}_i, \mathcal{LP}_i, \mathcal{MP}_i, \mathcal{IC}_i \rangle$, with $i \in [1..n]$, the set of stable models of $\mathcal{D}_i \cup \mathcal{LP}_i \cup \mathcal{IC}_i$ represents the

local knowledge of \mathcal{P}_i . We assume that each peer is *locally consistent*, that is such a set is not empty (i.e. $\mathcal{SM}(\mathcal{D}_i \cup \mathcal{LP}_i \cup \mathcal{IC}_i) \neq \emptyset$). A P2P system whose peers are locally consistent is *consistent*. Therefore, inconsistencies may be introduced just when a peer imports data from other peers. The intuitive meaning of a mapping rule $H \leftrightarrow \mathcal{B} \in \mathcal{MP}_i$ is that if the body conjunction \mathcal{B} is *true* in the source peer \mathcal{P}_j the atom H can be imported in \mathcal{P}_i only if it does not imply (directly or indirectly) the violation of some integrity constraint in \mathcal{IC}_i . The following example will clarify the meaning of mapping rules.

Example 2. Consider the P2P system in Fig. 1. If the fact 1: $p(b)$ is imported in \mathcal{P}_1 , the fact 1: $r(b)$ will be derived. As 1: $r(a)$ is already true in \mathcal{P}_1 , because it is derived from 1: $s(a)$, the integrity constraint is violated. Therefore, 1: $p(b)$ cannot be imported in \mathcal{P}_1 as it indirectly violates an integrity constraint. \square

Before formally presenting the preferred weak model semantics, some notation is introduced. Given a mapping rule $r = H \leftrightarrow \mathcal{B}$, the corresponding standard rule $H \leftarrow \mathcal{B}$ will be denoted as $St(r)$. Analogously, given a set of mapping rules \mathcal{MP} , $St(\mathcal{MP}) = \{St(r) \mid r \in \mathcal{MP}\}$ and given a P2P system $\mathcal{PS} = \mathcal{D} \cup \mathcal{LP} \cup \mathcal{MP} \cup \mathcal{IC}$, $St(\mathcal{PS}) = \mathcal{D} \cup \mathcal{LP} \cup St(\mathcal{MP}) \cup \mathcal{IC}$.

Given an interpretation M , an atom H and a conjunction of atoms \mathcal{B} :

- $val_M(H \leftarrow \mathcal{B}) = val_M(H) \geq val_M(\mathcal{B})$,
- $val_M(H \leftrightarrow \mathcal{B}) = val_M(H) \leq val_M(\mathcal{B})$.

Therefore, while a standard rule is satisfied if its body is *false* or its body is *true* and its head is *true*, a mapping rule is satisfied if its body is *true* or its body is *false* and its head is *false*.

Intuitively, a *weak model* M of a P2P system \mathcal{PS} is an interpretation that satisfies all standard rules, mapping rules and constraints of \mathcal{PS} and such that each atom $H \in M[\mathcal{MP}]$ (i.e. each mapping atom) is *supported* from a mapping rule $H \leftrightarrow \mathcal{B}$ whose body \mathcal{B} is satisfied by M . A *preferred weak model* is a weak model that contains a maximal subset of mapping atoms. This concept is justified by the assumption that it is *preferable* to import in each peer *as much knowledge as possible*.

Definition 3. [(PREFERRED) WEAK MODEL]. Given a P2P system $\mathcal{PS} = \mathcal{D} \cup \mathcal{LP} \cup \mathcal{MP} \cup \mathcal{IC}$, an interpretation M is a *weak model* for \mathcal{PS} if $\{M\} = \mathcal{MM}(St(\mathcal{PS}^M))$, where \mathcal{PS}^M is the program obtained from $ground(\mathcal{PS})$ by:

- removing all peer rules whose body is *false* w.r.t. M ;
- removing all mapping rules whose head is *false* w.r.t. M ;
- removing from the remaining rules each negative literal.

Given two weak models M and N , M is said to be *preferable* to N , and is denoted as $M \supseteq N$, if $M[\mathcal{MP}] \supseteq N[\mathcal{MP}]$. Moreover, if $M \supseteq N$ and $N \not\supseteq M$, then $M \sqsupset N$. A weak model M is said to be *preferred* if there is no weak model N such that $N \sqsupset M$. The set of weak models for a P2P system \mathcal{PS} will be denoted by $\mathcal{WM}(\mathcal{PS})$, whereas the set of preferred weak models will be denoted by $\mathcal{PWM}(\mathcal{PS})$. \square

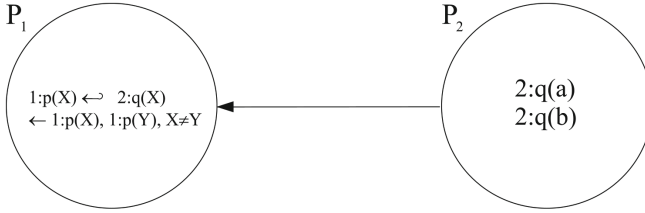


Fig. 2. The system \mathcal{PS}

It has been proved in our previous work that P2P systems always admit maximal weak models.

Theorem 1. For every consistent P2P system \mathcal{PS} , $\mathcal{PWM}(\mathcal{PS}) \neq \emptyset$.

Example 3. Consider the P2P system \mathcal{PS} in Fig. 2. \mathcal{P}_2 contains the facts $2:q(a)$ and $2:q(b)$, whereas \mathcal{P}_1 contains the mapping rule $1:p(X) \leftrightarrow 1:q(X)$ and the constraint $\leftarrow 1:p(X), 1:p(Y), X \neq Y$. The weak models of the system are $M_0 = \{2:q(a), 2:q(b)\}$, $M_1 = \{2:q(a), 2:q(b), 1:p(a)\}$ and $M_2 = \{2:q(a), 2:q(b), 1:p(b)\}$, whereas the preferred weak models are M_1 and M_2 as they import the maximal set of atoms from \mathcal{P}_2 . □

4 A More General Framework

We first provides the a definition of *P2P system* that generalizes Definition 2 by introducing a new type of peers - *the unsound peers* - giving more priority to imported data with respect to local data.

Definition 4. An *unsound P2P system* \mathcal{UPS} is a pair $(\mathcal{PS}, \mathcal{U})$, where $\mathcal{PS} = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ is a (standard) P2P system and $\mathcal{U} \subseteq \mathcal{PS}$. Peers in \mathcal{U} are called *unsound peers*. □

The semantics of an unsound P2P system \mathcal{UPS} is derived from the weak model semantics of a corresponding standard P2P system obtained from \mathcal{UPS} by splitting each unsound peer \mathcal{P}_i into two peers. The idea is to move the local database \mathcal{D}_i from \mathcal{P}_i to the new peer and to introduce in \mathcal{P}_i a set of special mapping rules able to import only portions of \mathcal{D}_i that do not violate its integrity constraints only after previously existing mapping rules imported data from the rest of the system.

Definition 5. Let $\mathcal{UPS} = (\mathcal{PS}, \mathcal{U})$ an unsound P2P system, where $\mathcal{PS} = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, and $\mathcal{P}_i = \langle \mathcal{D}_i, \mathcal{LP}_i, \mathcal{MP}_i, \mathcal{IC}_i \rangle$ a peer in \mathcal{U} . Then, $Split(\mathcal{P}_i)$ is the set containing the following peers:

- $\mathcal{P}_{(i+n)} := \langle \{(i+n):p(X) \mid i:p(X) \in \mathcal{D}_i\}, \emptyset, \emptyset, \emptyset \rangle$

- $\mathcal{P}_i := \langle \emptyset, \mathcal{LP}_i, \mathcal{MP}_i \cup \mathcal{MD}_i, \mathcal{IC}_i \rangle$, with $\mathcal{MD}_i = \{i:p(X) \leftrightarrow (n+i):p(X) \mid (n+i):p \text{ is a base predicate defined in } \mathcal{P}_{(i+n)}\}$.

Moreover:

$$Split(\mathcal{UPS}) = (\mathcal{PS} \setminus \mathcal{U}) \cup \bigcup_{\mathcal{P}_i \in \mathcal{U}} Split(\mathcal{P}_i)$$

□

With a little abuse of notations, given an unsound P2P system \mathcal{UPS} , $Split(\mathcal{UPS})$ will be also denoted as $\langle \mathcal{D}, \mathcal{LP}, \mathcal{MP} \cup \mathcal{MD}, \mathcal{IC} \rangle$. In previous definition, the peer \mathcal{P}_i is redefined by deleting its local database \mathcal{D}_i and inserting mapping rules \mathcal{MD} allowing to import tuples into old base relations (which now are mapping relations) from an auxiliary peer $\mathcal{P}_{(i+n)}$. This split is not visible to other peers as all predicates defined in \mathcal{P}_i are still defined.

Example 4. Let us continue our discussion about the P2P system \mathcal{PS} presented in Example 1. Assuming that \mathcal{PS} is an unsound P2P system and \mathcal{P}_1 is an unsound peer, $Split(\mathcal{PS})$ contains the following peers (see Fig. 3):

- $\mathcal{P}_1 = \langle \emptyset, \mathcal{LP}_1, \mathcal{MP}_1 \cup \{1:s(X) \leftrightarrow 3:s(X)\}, \mathcal{IC}_1 \rangle$
- $\mathcal{P}_2 = \langle \{2:q(b)\}, \emptyset, \emptyset, \emptyset \rangle$
- $\mathcal{P}_3 = \langle \{3:s(a)\}, \emptyset, \emptyset, \emptyset \rangle$

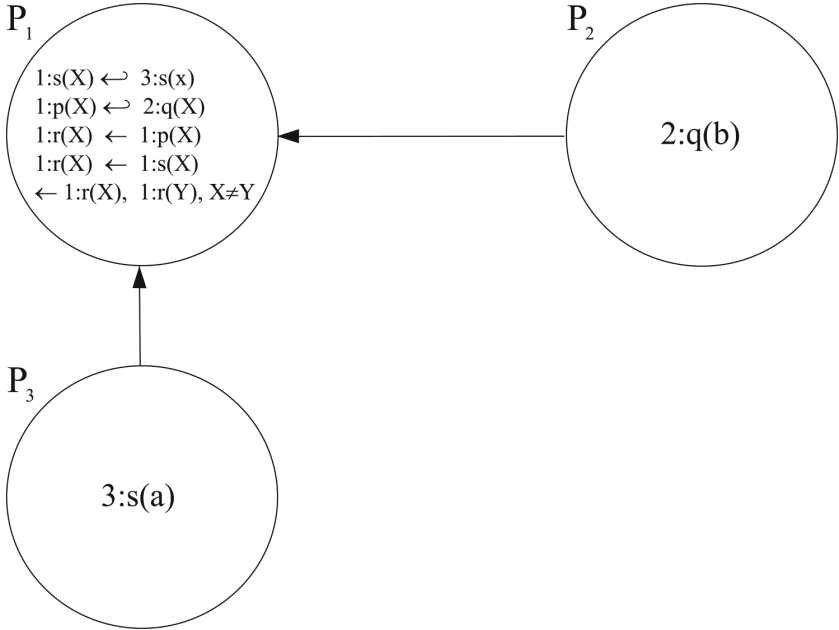


Fig. 3. The system \mathcal{PS}

□

We need to modify the definition of preferred weak model reported in Sect. 3.2 in order to ensure local data to be less preferred with respect to imported data. The new semantics of the unsound P2P system is given as follow:

Definition 6. [SEMANTICS] Let $UPS = (\mathcal{PS}, \mathcal{U})$ be an unsound P2P system and $Split(UPS) = \langle \mathcal{D}, \mathcal{LP}, \mathcal{MP} \cup \mathcal{MD}, \mathcal{IC} \rangle$.

For any two weak models M and N of $Split(UPS)$, we say that M is *preferable* to N ($M \sqsupseteq N$) if:

- $M[\mathcal{MP}] \supset N[\mathcal{MP}]$, or
- $M[\mathcal{MP}] = N[\mathcal{MP}]$ and $M[\mathcal{MD}] \supseteq N[\mathcal{MD}]$.

Moreover, if $M \sqsupseteq N$ and $N \not\sqsupseteq M$ we write $M \sqsupset N$. A weak model M is said to be *preferred* if there is no weak model N such that $N \sqsupset M$. We denote as $\mathcal{PWM}(UPS)$ the set of preferred weak models of UPS . \square

Example 5. Let us continue our discussion about the P2P system \mathcal{PS} presented in Example 1.

- Assuming that \mathcal{PS} is a sound P2P system, then \mathcal{P}_1 trusts its own data more than the data that can be imported from \mathcal{P}_2 . Therefore it will not import the fact $1:p(b)$ because it would violate its integrity constraint. The preferred weak models of \mathcal{PS} is: $M_1 = \{2:q(b), 1:s(a), 1:r(a)\}$
- Assuming that \mathcal{PS} is an unsound P2P system and \mathcal{P}_1 is an unsound peer we consider the peers in $Split(\mathcal{PS})$ reported in Example 4. In this case, \mathcal{P}_1 gives preference to the knowledge that can be imported from \mathcal{P}_2 with respect to its knowledge. Therefore, there is only one possible scenario: $1:p(b)$ is imported from \mathcal{P}_2 (i.e. it is derived from $2:q(b)$) and $1:s(a)$ is not imported from \mathcal{P}_3 (i.e. it is not derived from $3:s(a)$). This is equivalent to delete $1:s(a)$ from the original peer \mathcal{P}_1 . The corresponding *preferred weak model* is $\{2:q(b), 1:p(b), 1:r(b), 3:s(a)\}$. Observe that the absence of $1:s(a)$ together with the presence of $3:s(a)$ means that $1:s(a)$ has been deleted from the original peer \mathcal{P}_1 . \square

5 Query Answers and Complexity

We consider now the computational complexity of calculating preferred weak models and answers to queries for unsound P2P systems. As an unsound P2P system may admit more than one preferred weak model, the answer to a query is given by considering *brave* or *cautious* reasoning (also known as *possible* and *certain* semantics).

Definition 7. Given an unsound P2P system UPS and a ground peer atom A , then A is true under

- *brave reasoning* if $A \in \bigcup_{M \in \mathcal{PWM}(UPS)} M$,
- *cautious reasoning* if $A \in \bigcap_{M \in \mathcal{PWM}(UPS)} M$. \square

We assume here a simplified framework not considering the distributed complexity as we suppose that the complexity of communications depends on the number of computed atoms which are the only elements exported by peers.

Theorem 2. *Let UPS be an unsound P2P system, then:*

1. *Deciding whether an interpretation M is a preferred weak model of UPS is $coNP$ complete.*
2. *Deciding whether a preferred weak model for UPS exists is in Σ_2^P .*
3. *Deciding whether an atom A is true in some preferred weak model of UPS is Σ_2^P complete.*
4. *Deciding whether an atom A is true in every preferred weak model of UPS is Π_2^P complete.*

6 Related Works

In this paper we introduce the concept of sound and unsound peer. A sound peer trusts more its own data with respect to the data imported from the rest of the system, whereas an unsound peer trusts less its own data with respect to the data imported from the rest of the system. This concept implicitly introduces a preference in the way the integration process is performed.

This paper is placed among the works on semantic peer data management systems. Among the approaches that are close to ours, we mention [9, 10, 21]. In all them mapping rules are of import kind and none of them explicitly uses trust relationships. In [9, 10] a semantics for a P2P system, based on epistemic logic, is proposed. The paper proposes a sound, complete and terminating procedure that returns the certain answers to a query submitted to a peer. The advantage of this framework is that certain answers of fixed conjunctive queries posed on a peer can be computed in polynomial time. A peer that results to be inconsistent with respect to its local constraints is ignored. New atoms are imported in a peer by means of mapping rules if this ensures to maintain local consistency.

In [20–22] a characterization of P2P database systems and a model-theoretic semantics dealing with inconsistent peers is proposed. The basic idea is that if a peer does not have models all (ground) queries submitted to the peer are *true* (i.e. are *true* with respect to all models). Thus, if some databases are inconsistent it does not mean that the entire system is inconsistent. The semantics in [21] coincides with the epistemic semantics in [9, 10].

Interesting semantics for data exchange systems that offer the possibility of modeling some preference criteria while performing the data integration process has been proposed in [4–6, 15, 16]. In [4–6] it is proposed a new semantics that allows for a cooperation among pairwise peers that related each other by means of data exchange constraints (i.e. mapping rules) and trust relationships. The decision by a peer on what other data to consider (besides its local data) does not depend only on its mapping rules, but also on the trust relationship that it has with other peers. Given a peer \mathcal{P} in a P2P system a solution for \mathcal{P} is a database instance that respects the mapping rules and trust relationship \mathcal{P} has with its

‘immediate neighbors’. Trust relationships are of the form: $(\mathcal{P}, less, \mathcal{Q})$ stating that \mathcal{P} trusts itself less than \mathcal{Q} , and $(\mathcal{P}, same, \mathcal{Q})$ stating that \mathcal{P} trusts itself the same as \mathcal{Q} . These trust relationships are static and are used in the process of collecting data in order to establish preferences in the case of conflicting information.

In [15] it is defined a mechanism that allows to set different degree of reliability for neighbor peers.

Both in [15] and in [4,6] the mechanism is *rigid* as the preference among conflicting sets of atoms only depends on the priorities (trust relationship) fixed at design time. In order to overcome static preferences, in [16] ‘dynamic’ preferences are introduced. They allow to select among different scenarios looking at the properties of provided data. This allows modeling concepts like “*in the case of conflicting information, it is preferable to import data from the neighbor peer that can provide the maximum number of tuples*” without selecting a-priori preferred peers.

In [12] the basic spirit of P2P system has been followed and a semantics that in which a peer gives no preference to its knowledge with respect to the knowledge that can be imported from other peers is presented.

7 Conclusion

The paper is placed among the works on semantic peer data management systems. It introduces the concept of sound and unsound peer: a sound peer trusts more its own data with respect to the data imported from the rest of the system, whereas an unsound peer trusts less its own data with respect to the data imported from the rest of the system. This concept implicitly introduces a preference in the way the integration process is performed. The paper introduces a logic programming based framework and proposes a new semantics capturing this simple idea.

As for future works, we plan to extend the framework in order to cope with a finer granularity of self esteem for a generic peer. In addition this concept, could be combined with mechanism that allows to set different degree of reliability for neighbor peers, in a way similar to the proposal in [15]. Another possible extension consists in express the self esteem of the relation of a peer not of the entire peer as we have done in this work and in the work in [16].

References

1. Abiteboul, S., Duschka, O.M.: Complexity of answering queries using materialized views. In: ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems. PODS 1998, pp. 254–263 (1998)
2. Arenas, M., Bertossi, L., Chomicki, J.: Consistent query answers in inconsistent databases. In: Symposium on Principles of Database Systems, pp. 68–79 (1999)
3. Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayen, I.: Data management for peer-to-peer computing: a vision. In: WebDB, pp. 89–94 (2002)

4. Bertossi, L., Bravo, L.: Query answering in peer-to-peer data exchange systems. In: Lindner, W., Mesiti, M., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) *Current Trends in Database Technology*. LNCS, vol. 3568, pp. 476–485. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30192-9_47
5. Bertossi, L., Bravo, L.: The semantics of consistency and trust in peer data exchange systems. In: Dershowitz, N., Voronkov, A. (eds.) *LPAR 2007*. LNCS (LNAI), vol. 4790, pp. 107–122. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75560-9_10
6. Bertossi, L., Bravo, L.: Consistency and trust in peer data exchange systems. *TPLP* **17**(2), 148–204 (2017)
7. Cali, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: On the decidability and complexity of query answering over inconsistent and incomplete databases. In: *PODS*, pp. 260–271 (2003)
8. Calvanese, D., Damaggio, E., De Giacomo, G., Lenzerini, M., Rosati, R.: Semantic data integration in P2P systems. In: Aberer, K., Koubarakis, M., Kalogeraki, V. (eds.) *DBISP2P 2003*. LNCS, vol. 2944, pp. 77–90. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24629-9_7
9. Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Logical foundations of peer-to-peer data integration. In: *PODS*, pp. 241–251 (2004)
10. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Inconsistency tolerance in P2P data integration: an epistemic logic approach. *Inf. Syst.* **33**(4–5), 360–384 (2008)
11. Caroprese, L., Greco, S., Zumpano, E.: A logic programming approach to querying and integrating P2P deductive databases. In: *The International Florida AI Research Society Conference*, pp. 31–36 (2006)
12. Caroprese, L., Zumpano, E.: A declarative semantics for P2P systems. In: Holzinger, A., Kieseberg, P., Tjoa, A.M., Weippl, E. (eds.) *CD-MAKE 2017*. LNCS, vol. 10410, pp. 315–329. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66808-6_21
13. Caroprese, L., Molinaro, C., Zumpano, E.: Integrating and querying P2P deductive databases. In: *International Database Engineering & Applications Symposium*, pp. 285–290 (2006)
14. Caroprese, L., Zumpano, E.: Consistent data integration in P2P deductive databases. In: Prade, H., Subrahmanian, V.S. (eds.) *SUM 2007*. LNCS (LNAI), vol. 4772, pp. 230–243. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75410-7_17
15. Caroprese, L., Zumpano, E.: Modeling cooperation in P2P data management systems. In: An, A., Matwin, S., Raś, Z.W., Ślęzak, D. (eds.) *ISMIS 2008*. LNCS (LNAI), vol. 4994, pp. 225–235. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68123-6_25
16. Caroprese, L., Zumpano, E.: Aggregates and priorities in P2P data management systems. In: *IDEAS 2011*, pp. 1–7 (2011)
17. Caroprese, L., Zumpano, E.: Handling preferences in P2P systems. In: Lukasiewicz, T., Sali, A. (eds.) *FoIKS 2012*. LNCS, vol. 7153, pp. 91–106. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28472-4_6
18. Caroprese, L., Zumpano, E.: Generalized maximal consistent answers in P2P deductive databases. In: Hartmann, S., Ma, H. (eds.) *DEXA 2016 Part II*. LNCS, vol. 9828, pp. 368–376. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44406-2_30
19. Fagin, R., Kolaitis, P.G., Popa, L.: Data exchange: getting to the core. *ACM Trans. Database Syst.* **30**(1), 174–210 (2005)

20. Franconi, E., Kuper, G.M., Lopatenko, A., Zaihrayeu, I.: Queries and updates in the coDB peer to peer database system. In: VLDB, pp. 1277–1280 (2004)
21. Franconi, E., Kuper, G.M., Lopatenko, A., Zaihrayeu, I.: A robust logical and computational characterisation of pecto-peer database systems. In: DBISP2P 2003, pp. 64–76 (2003)
22. Franconi, E., Kuper, G.M., Lopatenko, A., Zaihrayeu, I.: A distributed algorithm for robust data sharing and updates in P2P database networks. In: Lindner, W., Mesiti, M., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) EDBT 2004, vol. 3268, pp. 446–455. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30192-9_44
23. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: ICLP/SLP 1988, pp. 1070–1080 (1988)
24. Greco, G., Greco, S., Zumpano, E.: Repairing and querying inconsistent databases. *Trans. Knowl. Data Eng.* **15**, 1389–1408 (2003)
25. Halevy, A., Ives, Z.G., Suci, D., Tatarinov, I.: Schema mediation in peer data management systems. In: ICDT, pp. 505–516 (2003)
26. Halevy, A.Y., Ives, Z.G., Suci, D., Tatarinov, I.: Schema mediation for large-scale semantic data sharing. *VLDB J.* **14**(1), 68–83 (2005)
27. Lenzerini, M.: Data integration: a theoretical perspective. In: PODS, pp. 233–246 (2002)
28. Madhavan, J., Halevy, A.Y.: Composing mappings among data sources. In: VLDB, pp. 572–583 (2003)
29. Tatarinov, I., Halevy, A.: Efficient query reformulation in peer data management systems. In: SIGMOD, pp. 539–550 (2004)

Author Index

- Abelló, Alberto 200
Arolfo, Franco 159
- Baazaoui Zghal, Hajer 247
Bikakis, Nikos 50
Borges, Felipe Ferreira 216
Broneske, David 113
- Campero, Gabriel 113
Caroprese, Luciano 278
Cordeiro, Robson L. F. 142
Cuzzocrea, Alfredo 216
- Del Fabro, Marcos Didonet 173
do Carmo Oliveira, Diego Junior 216
- Ferrarotti, Flavio 99
Ferri, Junior 173
Ficel, Hemza 247
- González, Senén 99
Gurumurthy, Bala 113
- Haddad, Mohamed Ramzi 247
Hagedorn, Stefan 185
Hai, Rihan 35
- Kaster, Daniel S. 142
Klabjan, Diego 82
Kuralenok, Igor E. 233
- Lehner, Wolfgang 200
- Mancas, Christian 263
Maroulis, Stavros 50
Marshalkin, Nikita 233
- Mic, Vladimir 127
Munir, Rana Faisal 200
- Novak, David 127
Novikov, Boris 233
- Papastefanatos, George 50
Papp, Dávid 69
Pinnecke, Marcus 113
- Quix, Christoph 35
- Revesz, Peter Z. 20
Ribeiro, Leonardo Andrade 216
Romero, Oscar 200
- Saake, Gunter 113
Sattler, Kai-Uwe 185
Schewe, Klaus-Dieter 99
Szalay, Alexander S. 3
Szűcs, Gábor 69
- Teng, Xu 82
Thiele, Maik 200
Tissot, Hegler 173
Trajcevski, Goce 82
Trofimov, Artem 233
- Vadicamo, Lucia 127
Vaisman, Alejandro 159
Vasconcelos, Guilherme Q. 142
Vassiliadis, Panos 50
- Zezula, Pavel 127
Zhou, Chen 35
Züfle, Andreas 82
Zumpano, Ester 278