Georg Gottlob
Giovanni Grasso
Dan Olteanu
Christian Schallhart (Eds.)

LNCS 7968

Big Data

29th British National Conference on Databases, BNCOD 2013
Oxford, UK, July 2013
Proceedings

Springer

Lecture Notes in Computer Science 7968

Georg Gottlob   Giovanni Grasso
Dan Olteanu   Christian Schallhart (Eds.)

Big Data

29th British National Conference
on Databases, BNCOD 2013
Oxford, UK, July 8-10, 2013
Proceedings

Springer

Volume Editors

Georg Gottlob
Giovanni Grasso
Dan Olteanu
Christian Schallhart

University of Oxford
Wolfson Building, Parks Road
Oxford, OX1 3QD, UK

E-mail:
{georg.gottlob; giovanni.grasso; dan.olteanu; christian.schallhart}@cs.ox.ac.uk

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-39466-9 e-ISBN 978-3-642-39467-6
DOI 10.1007/978-3-642-39467-6
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013942451

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

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

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

Printed on acid-free paper

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

Preface

This volume contains the papers presented at BNCOD 2013: 29th British National Conference on Databases held during July 7–9, 2013, in Oxford.

The BNCOD Conference is a venue for the presentation and discussion of research papers on a broad range of topics related to data-centric computation. For some years, every edition of BNCOD has centered around a main theme, acting as a focal point for keynote addresses, tutorials, and research papers. The theme of BNCOD 2013 is *Big Data*. It encompases a growing need to manage data that is too big, too fast, or too hard for the existing technology.

This year, BNCOD attracted 42 complete submissions from 14 different African, European, South and North American countries. Each submission was reviewed by three Program Committee members. The committee decided to accept 20 papers on such topics as query and update processing, relational storage, benchmarking, XML query processing, Big Data, spatial data, indexing, data extraction and social networks. The conference program also included three keynote talks, two tutorials, and one panel session.

We would like to thank the authors for supporting BNCOD by submitting their work to the conference, the Program Committee members for their help in shaping an excellent conference program, and the distinguished speakers for accepting our invitation. Thanks also go to Elizabeth Walsh, Karen Barnes, and Andrea Pilot for their involvement in the local organization of the conference.

May 2013

Georg Gottlob
Giovanni Grasso
Dan Olteanu
Christian Schallhart

Organization

Program Committee

Marcelo Arenas	Pontificia Universidad Catolica de Chile, Chile
François Bry	Ludwig Maximilian University, Munich, Germany
Andrea Calì	Birkbeck College, University of London, UK
James Cheney	University of Edinburgh, UK
Thomas Eiter	TU Vienna, Austria
Tim Furche	University of Oxford, UK
Floris Geerts	University of Antwerp, Belgium
Jun Hong	Queen's University Belfast, UK
Stratos Idreos	CWI Amsterdam, The Netherlands
Mike Jackson	Birmingham City University, UK
Anne James	Coventry University, UK
Georg Lausen	Albert Ludwig University, Freiburg, Germany
Lachlan Mackinnon	University of Greenwich, UK
Sebastian Maneth	University of Oxford, UK
Peter McBrien	Imperial College, London, UK
David Nelson	University of Sunderland, UK
Werner Nutt	Free University of Bozen-Bolzano, Italy
Dan Olteanu	University of Oxford, UK
Norman Paton	University of Manchester, UK
Reinhard Pichler	TU Vienna, Austria
Mark Roantree	Dublin City University, Ireland
Florin Rusu	University of California, Merced, USA
Sandra Sampaio	University of Manchester, UK
Pierre Senellart	Telecom Paris Tech, France
Letizia Tanca	Politecnico di Milano, Italy
Andy Twigg	University of Oxford, UK
Stratis Viglas	University of Edinburgh, UK
John Wilson	University of Strathclyde, UK
Peter Wood	Birkbeck College, University of London, UK

Additional Reviewers

Antova, Lyublena
Fink, Robert
Mazuran, Mirjana
Pieris, Andreas
Quintarelli, Elisa
Savenkov, Vadim
Schallhart, Christian
Schneider, Patrik
Simkus, Mantas
Spina, Cinzia Incoronata
Xiao, Guohui
Zavodny, Jakub

Table of Contents

Keynotes

Tutorials

Panel

Query and Update Processing

Relational Storage

Benchmarking

XML Query Processing

Big Data

Spatial Data and Indexing

Data Extraction and Social Networks

Big Data Begets Big Database Theory*

Dan Suciu

University of Washington

1 Motivation

Industry analysts describe Big Data in terms of three V's: volume, velocity, variety. The data is too big to process with current tools; it arrives too fast for optimal storage and indexing; and it is too heterogeneous to fit into a rigid schema. There is a huge pressure on database researchers to study, explain, and solve the technical challenges in big data, but we find no inspiration in the three Vs. Volume is surely nothing new for us, streaming databases have been extensively studied over a decade, while data integration and semistructured has studied heterogeneity from all possible angles.

So what makes Big Data special and exciting to a database researcher, other for the great publicity that our field suddenly gets? This talk argues that the novelty should be thought along different dimensions, namely in communication, iteration, and failure.

Traditionally, database systems have assumed that the main complexity in query processing is the number of disk IOs, but today that assumption no longer holds. Most big data analysis simply use a large number of servers to ensure that the data fits in main memory: the new complexity metric is the amount of communication between the processing nodes, which is quite novel to database researchers.

Iteration is not that new to us, but SQL has adopted iteration only lately, and only as an afterthought, despite amazing research done on datalog in the 80s [1]. But Big Data analytics often require iteration, so it will play a center piece in Big Data management, with new challenges arising from the interaction between iteration and communication [2].

Finally, node failure was simply ignored by parallel databases as a very rare event, handled with restart. But failure is a common event in Big Data management, when the number of servers runs into the hundreds and one query may take hours [3].

The Myria project [4] at the University of Washington addresses all three dimensions of the Big Data challenge. Our premise is that each dimension requires a study of its fundamental principles, to inform the engineering solutions. In this talk I will discuss the communication cost in big data processing, which turns out to lead to a rich collection of beautiful theoretical questions; iteration and failure are left for future research.

* This work was partially supported by NSF IIS-1115188, IIS-0915054 and IIS-1247469.

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 1–5, 2013.

2 The Question

Think of a complex query on a big data. For example, think of a three-way join followed by an aggregate, and imagine the data is distributed on one thousand servers. How many communication rounds are needed to compute the query? Each communication round typically requires a complete reshuffling of the data across all 1000 nodes, so it is a very expensive operation, we want to minimize the number of rounds. For example, in MapReduce [5], a MR job is defined by two functions: `map` defines how the data is reshuffled, and `reduce` performs the actual computation on the repartitioned data. A complex query requires several MR jobs, and each job represents one global communication round. We can rephrase our question as: how many MR jobs are necessary to compute the given query? Regardless of whether we use MR or some other framework, fewer communication rounds mean less data exchanged, and fewer global synchronization barriers. The fundamental problem that we must study is: determine the minimum number of global communication rounds required to compute a given query.

3 The Model

MapReduce is not suitable at all for theoretical lower bounds, because it allows us to compute *any* query in *one* round: simply map all data items to the same intermediate key, and perform the entire computation sequentially, using one reducer. In other words, the MR model does not prevent us from writing a sequential algorithm, and it is up to the programmer to avoid that by choosing a sufficiently large number of reducers.

Instead, we consider the following simple model, called the Massively Parallel Communication (MPC) model, introduced in [6]. There are a fixed number of servers, p, and the input data of size n is initially uniformly distributed on the servers; thus, each server holds $O(n/p)$ data items. The computation proceeds in rounds, where each round consists a computation step and a global communication step. The communication is many-to-many, allowing a total reshuffling of the data, but with the restriction that each server receives only $O(n/p)$ amount of data. The servers have unrestricted computational power, and unlimited memory: but because of the restriction on the communication, after a constant number of rounds, each server sees only a fraction $O(n/p)$ of the input data. In this model we ask the question: given a query, how many rounds are needed to compute it? A naive solution that sends the entire data to one server is now ruled out, since the server can only receive $O(n/p)$ of the data.

A very useful relaxation of this model is one that allows each server to receive $O(n/p \times p^{\varepsilon})$ data items, where $\varepsilon \in [0, 1]$. Thus, during one communication round the entire data is replicated by a factor p^{ε}; we call ε the *space exponent*. The case $\varepsilon = 0$ corresponds to the base model, while the case $\varepsilon = 1$ is uninteresting, because it allows us to send the entire data to every server, like in the MapReduce example.

4 The Cool Example

Consider first computing a simple join: $q(x, y, z) = R(x, y), S(y, z)$. This can be done easily in one communication round. In the first step, every server inspects its fragment of R, and sends every tuple of the form $R(a_1, b_1)$ to the destination server with number $h(b_1)$, where h is a hash function returning a value between 1 and p; similarly, it sends every tuple $S(b_2, c_2)$ to server $h(b_2)$. After this round of communication, the servers compute the join locally and report the answers.

A much more interesting example is $q(x, y, z) = R(x, y), S(y, z), T(z, y)$. When all three symbols R, S, T denote the same relation, then the query computes all triangles in the data[1], a popular task in Big Data analysis [7]. Obviously, this query can be computed in two communication rounds, doing two separate joins. But, quite surprisingly, it can be computed in a single communication round! The underlying idea has been around for 20 years [8,9,7], but, to our knowledge, has not yet been deployed in practice. We explain it next.

To simplify the discussion, assume $p = 1000$ servers. Then each server can be uniquely identified as a triple (i, j, k), where $1 \leq i, j, k \leq 10$. Thus, the servers are arranged in a cube, of size $10 \times 10 \times 10$. Fix three independent hash functions h_1, h_2, h_3, each returning values between 1 and 10. During the single communication round, each server sends the following:

- $R(a_1, b_1)$ to the servers $(h_1(a_1), h_2(b_1), 1), \ldots, (h_1(a_1), h_2(b_1), 10)$
- $S(b_2, c_2)$ to the servers $(1, h_2(b_2), h_3(c_2)), \ldots, (10, h_2(b_2), h_3(c_2))$
- $T(c_3, a_3)$ to the servers $(h_1(a_3), 1, h_3(c_3)), \ldots, (h_1(a_3), 10, h_3(c_3))$

In other words, when inspecting $R(a_1, b_1)$ a server can compute the i and j coordinates of the destination (i, j, k), but doesn't know the k coordinate, and it simply replicates the tuple to all 10 servers. After this communication step, every server computes locally the triangles that it sees, and reports them. The algorithm is correct, because every potential triangle (a, b, c) is seen by some server, namely by $(h_1(a), h_2(b), h_3(c))$. Moreover, the data is replicated only 10 times. The reader may check that, if the number of servers is some arbitrary number p, then the amount of replication is $p^{1/3}$, meaning that the query can be computed in one communication round using a space exponent $\varepsilon = 1/3$.

It turns out that $1/3$ is the *smallest* space exponent for which we can compute q in one round! Moreover, a similar result holds for every conjunctive query without self-joins, as we explain next.

5 Communication Complexity in Big Data

Think of a conjunctive query q as a hypergraph. Every variable is a node, and every atom is an hyperedge, connecting the variables that occur in that atom. For our friend $R(x, y), S(y, z), T(z, y)$ the hypergraph is a graph with three nodes

[1] The query $q(x, y, z) = R(x, y), R(y, z), R(z, x)$ reports each triangle three times; to avoid double counting, one can modify the query to $q(x, y, z) = R(x, y), R(y, z), R(z, x), x < y < z$.

x, y, z and three edges, denoted R, S, T, forming a triangle. We will refer interchangeably to a query as a hypergraph.

A *vertex cover* of a query q is a subset of nodes such that every edge contains at least one node in the cover. A *fractional vertex cover* associates to each variable a number ≥ 0 such that for each edge the sum of the numbers associated to its variables is ≥ 1. The value of the fractional vertex cover is the sum of the numbers of all variables. The smallest value of any fractional vertex cover is called the *fractional vertex cover* of q and is denoted $\tau^*(q)$.

For example, consider our query $q(x, y, z) = R(x, y), S(y, z), T(z, y)$. Any vertex cover must include at least two variables, e.g. $\{x, y\}$, hence its value is 2. The fractional vertex cover $1/2, 1/2, 1/2$ (the numbers correspond to the variables x, y, z) has value $3/2$ and the reader may check that this is the smallest value of any fractional vertex cover; thus, $\tau^*(q) = 3/2$. The smallest space exponent needed to compute a query in one round is given by:

Theorem 1. *[6] If $\varepsilon < 1 - 1/\tau^*(q)$ then the query q cannot be computed in a single round on the MPC model with space exponent ε.*

To see another example, fix some $k \geq 2$ and consider a chain query $L_k = R_1(x_0, x_1), R_2(x_1, x_2), \ldots, R_k(x_{k-1}, x_k)$. Its optimal fractional vertex cover is $0, 1, 0, 1, \ldots$ where the numbers correspond to the variables $x_0, x_1, x_2, \ldots$ Thus, $\tau^*(L_k) = \lceil k/2 \rceil$, and the theorem says that L_k cannot be computed with a space exponent $\varepsilon < 1 - 1/\lceil k/2 \rceil$.

What about multiple rounds? For a fixed $\varepsilon \geq 0$, let $k_\varepsilon = 2\lfloor 1/(1-\varepsilon) \rfloor$; this is the longest chain L_{k_ε} computable in one round given the space exponent ε. Let $\text{diam}(q)$ denote the diameter of the hypergraph q. Then:

Theorem 2. *[6] For any $\varepsilon \geq 0$, the number of rounds needed to compute q within a space exponent ε is at least $\lceil \log_{k_\varepsilon}(diam(q)) \rceil$.*

6 The Lessons

An assumption that has never been challenged in databases is that we always compute one join at a time[2]: a relational plan expresses the query as a sequence of simple operations, where each operation is either unary (group-by, selection, etc), or is binary (join). We never use plans with other than unary or binary operators, and never compute a three-way join directly. In Big Data this assumption needs to be revisited. By designing algorithms for multi-way joins, we can reduce the total communication cost for the query. The theoretical results discussed here show that we must also examine query plans with complex operators, which compute an entire subquery in one step. The triangle query is one example: computing it in one step requires that every table be replicated $p^{1/3}$ times (e.g. 10 times, when $p = 1000$), while computing it in two steps requires reshuffling the intermediate result $R(x, y), S(y, z)$, which, on a graph like twitter's `Follows` relation, is significantly larger than ten times the input table.

[2] We are aware of one exceptions: the Leap Frog join used by LogicBlox [10].

References

1. Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about datalog (and never dared to ask). IEEE Trans. Knowl. Data Eng. 1(1), 146–166 (1989)
2. Bu, Y., Howe, B., Balazinska, M., Ernst, M.D.: The haloop approach to large-scale iterative data analysis. VLDB J. 21(2), 169–190 (2012)
3. Upadhyaya, P., Kwon, Y., Balazinska, M.: A latency and fault-tolerance optimizer for online parallel query plans. In: SIGMOD Conference, pp. 241–252 (2011)
4. Myria, `http://db.cs.washington.edu/myria/`
5. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: OSDI, pp. 137–150 (2004)
6. Beame, P., Koutris, P., Suciu, D.: Communication steps for parallel query processing. In: PODS (2013)
7. Suri, S., Vassilvitskii, S.: Counting triangles and the curse of the last reducer. In: WWW, pp. 607–614 (2011)
8. Ganguly, S., Silberschatz, A., Tsur, S.: Parallel bottom-up processing of datalog queries. J. Log. Program. 14(1&2), 101–126 (1992)
9. Afrati, F.N., Ullman, J.D.: Optimizing joins in a map-reduce environment. In: EDBT, pp. 99–110 (2010)
10. Veldhuizen, T.L.: Leapfrog triejoin: a worst-case optimal join algorithm. CoRR abs/1210.0481 (2012)

Compilation and Synthesis in Big Data Analytics

Christoph Koch

École Polytechnique Fédérale De Lausanne

Abstract. Databases and compilers are two long-established and quite distinct areas of computer science. With the advent of the big data revolution, these two areas move closer, to the point that they overlap and merge. Researchers in programming languages and compiler construction want to take part in this revolution, and also have to respond to the need of programmers for suitable tools to develop data-driven software for data-intensive tasks and analytics. Database researchers cannot ignore the fact that most big-data analytics is performed in systems such as Hadoop that run code written in general-purpose programming languages rather than query languages. To remain relevant, each community has to move closer to the other. In the first part of this keynote, I illustrate this current trend further, and describe a number of interesting and inspiring research efforts that are currently underway in these two communities, as well as open research challenges. In the second part, I present a number of research projects in this space underway in my group at EPFL, including work on the static and just-in-time compilation of analytics programs and database systems, and the automatic synthesis of out-of-core algorithms that efficiently exploit the memory hierarchy.

Curriculum Vitae

Christoph Koch is a professor of Computer Science at EPFL, specializing in data management. Until 2010, he was an Associate Professor in the Department of Computer Science at Cornell University. Previously to this, from 2005 to 2007, he was an Associate Professor of Computer Science at Saarland University. Earlier, he obtained his PhD in Artificial Intelligence from TU Vienna and CERN (2001), was a postdoctoral researcher at TU Vienna and the University of Edinburgh (2001-2003), and an assistant professor at TU Vienna (2003-2005). He obtained his Habilitation degree in 2004. He has won Best Paper Awards at PODS 2002, ICALP 2005, and SIGMOD 2011, a Google Research Award (in 2009), and an ERC Grant (in 2011). He (co-)chaired the program committees of DBPL 2005, WebDB 2008, and ICDE 2011, and was PC vice-chair of ICDE 2008 and ICDE 2009. He has served on the editorial board of ACM Transactions on Internet Technology as well as in numerous program committees. He currently serves as PC co-chair of VLDB 2013 and Editor-in-Chief of PVLDB.

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, p. 6, 2013.

The Providence of Provenance

Peter Buneman

School of Informatics,
University of Edinburgh

Abstract. For many years and under various names, provenance has been modelled, theorised about, standardised and implemented in various ways; it has become part of mainstream database research. Moreover, the topic has now infected nearly every branch of computer science: provenance is a problem for everyone. But what exactly is the problem? And has the copious research had any real effect on how we use databases or, more generally, how we use computers.

This is a brief attempt to summarise the research on provenance and what practical impact it has had. Although much of the research has yet to come to market, there is an increasing interest in the topic from industry; moreover, it has had a surprising impact in tangential areas such as data integration and data citation. However, we are still lacking basic tools to deal with provenance and we need a culture shift if ever we are to make full use of the technology that has already been developed.

1 Why Were We Interested?

It is well over 20 years since the issue of provenance was first introduced to, and initially ignored by, the database community [28]. It is noteworthy that much of the early work on this topic [28,12,9] was initiated by the need to understand provenance in the context of data integration. In fact, the earliest [28] paper states this very well:

> although the users want the simplicity of making a query as if it were a single large database, they also want the ability to know the source of each piece of data retrieved.

The other early paper on the topic [29] was more concerned with scientific programming, but even this addressed the provenance of "each piece of data" in an array. My own stimulus for studying provenance came from some molecular biologists with whom we were collaborating on data integration. The problem was compounded by the fact that they were not just building a data warehouse but also manually correcting and augmenting the curated database they were publishing.

Although database queries are relatively simple programs – they do little more than rearrange the data from their input – the "source of each piece of data retrieved" turns out to be a rather elusive concept. At least there are numerous ways of describing the source: one may ask *why* a tuple is in the output [12,9] or

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 7–12, 2013.

how it was constructed [19,21]. Even the description of *where* a "piece of data" has been copied from is non-trivial [7].

At the same time that the initial research on provenance and database queries was taking off, computer science researchers involved in scientific workflows started to think about provenance, and now several establish workflow systems provide some form of provenance capture [30,5,4,13,4,22]. Anyone who has done any large-scale, complex, scientific programming, such as data analysis or simulation will know how easy it is to lose track of one's efforts. One wants to be able to repeat the work, and this calls for some form of provenance. Here we are in a different playing-field. First, we are interested in the whole output, not one "piece of data"; second, the subroutines that one invokes may be much more complicated than database queries; and, finally, one may not trust those subroutines. To elaborate on this last point, database query languages have precise specifications, and we expect them to perform according to those specifications so we may not need to record the details of their execution. This is not the case with scientific programs in general.

To deal with this, and perhaps to deal with other concepts in provenance – such as those arising from traditional provenance associated with historical artifacts – a W3C working group has produced a series of models of provenance [23,17]. Starting with a simple causal model of provenance, the model has been enriched with a variety of notions that describe various forms of interaction between processes and artifacts. For whatever reason, there has been little attempt to connect these models of provenance, which sometimes go under the name of "workflow provenance" with the languages e.g., the workflow languages, that produce provenance graphs. And while there are some obvious and basic connections between workflow and data provenance, there may be much more to be done in this area.

2 What Have We Achieved?

Possibly the greatest achievement has been the drawing together of a number of research communities to tackle what, at first sight, is a common problem. At a recent provenance workshop [1] there were people from the semantic Web, databases, programming languages, computer systems, scientific programming as well as curators who think about provenance in its traditional sense. In addition, the study of provenance has led to some solid technical connections. Program slicing, which attempts – for the purposes of debugging – to provide a part of trace of the execution of a program that is relevant to some part of the output, is closely connected with provenance and database queries [11]. In a series of papers [19,21,1] Tannen and his colleagues show how many of the extensions to relational databases: c-tables, probabilistic databases, multiset semantics, as well as a number of provenance models have a common and powerful generalization as an abstract "provenance polynomial" associated with each tuple in the answer

[1] TaPP'11. 3rd Usenix Workshop on the Theory and Practice of Provenance. Heraklion, June 2011.

to a query. Moreover, formalisms developed for provenance have influenced other areas such as data integration [18] and hypothetical query answering [15].

The W3C working groups have also achieved some form of consensus on a general model of provenance [23,17]. One of the things that is discussed in this work but has not yet been fully formalised is the notion of *granularity.* Provenance graphs should be capable of expansion into an increasingly fine-grained description and possibly connect with ideas of data provenance. This does not appear to be difficult, but the details do need to be worked out.

This process of repeated expansion of provenance raises the important question: How much provenance we should record? Why not record the workflow specification or the source code together with the input and re-execute the process if we want to know what happened? If we look at data provenance where the programs are queries, the details of "what happened" are typically not of interest. Database queries are declarative, and we assume they have performed according to the well-defined specification. As we have already noted, not all programs have a well-defined specification and those that do may contain errors. And, of course, if the purpose of keeping provenance is to "instrument" the execution of a program in order to understand performance [6,24], then the more detail the better.

In practice, we generally know what parts of a workflow are error-prone or are the cause of a performance bottleneck, so why not keep provenance for just those components, and ignore it for those that are properly specified? In doing the latter, there is, of course, an interesting interaction with privacy in workflows [14].

3 What Impact Have We Had?

Those of us who have been working on provenance for some time might pose the alternative question with: What impact did we expect it to have? It would be great if we could say that a major disaster was avoided because we kept provenance, and we probably can say this. The excellent version control systems that are used in software development already keep a form of provenance, and their use has almost certainly avoided a major software disaster, but we don't find this remarkable. In a related lecture, Margo Selzer [26] rates the impact of provenance research as zero – at least in the sense that it has not yet produced any headline-grabbing results.

Are we expecting too much? From my own experience, the recording of provenance information has sometimes made life easier for scientists but has seldom enabled them to do things that they could not, by more laborious means, already do. Here are two examples.

– In working with the IUPHAR database [27] we installed an archiving system [8] that recorded the complete history of the database. Proper archiving is an essential precursor of any provenance or citation system that involves an evolving database. One of the properties of the system, which was intended as a compression technique, is that it allows one to see the evolution of some *part* of the database. This was useful to the curators who did not have to open up, and search, old versions of the database in order to see how some table had evolved.

- There was a recent case [2] of an influential paper, which justifies the case for economic austerity measures being found by a student, who had been asked to reproduce the results, to contain significant errors. It required a lot of work by the student, who ultimately had to consult the authors, to do this. One could imagine that had the paper been "provenance enabled", or – better – "executable" [25,16], the student would have spotted the mistake immediately. Indeed the mistake might never have happened!

So perhaps the real impact of provenance is not so much in the tools and models that we are directly developing to deal with it, but in the ancillary work on archiving, data citation, annotation, executable papers, program slicing, etc. that are directly connected with it and that are being informed, to some extent, by those models and tools.

4 A Change of Attitude Is Needed

Although the scientific programming community is, of necessity, becoming aware of the need to record provenance in some form. It is unclear that other communities, such as the Semantic Web, have developed the same level of awareness. Moreover, most of us are blissfully unaware that we are constantly throwing away provenance information in our ordinary work. To take one example the "LOD cloud" [20] – a distributed collection of billions of RDF triples – was created mostly by transforming and linking existing data sets. Somewhere in this linkage you will find the data extracted from the CIA World Factbook [3], which gives the population of Afghanistan as 31,889,923. If you go the the CIA World Factbook [10], you will find the population of Afghanistan as 30,419.928, with an annotation indicating that this estimate was made in 2012 and a further annotation indicating that this is significantly different from a previous estimate of 33,809,937[2] Presumably the LOD figure was copied from an older version of the Factbook, but the version (provenance) is not obviously recorded and, more importantly, whatever annotations there were in the original have "fallen off" in the process that created the relevant LOD triples.

It is a non-trivial challenge to make this process of transforming and linking data "provenance aware", it appeard to require a substantial modification or even a complete restructuring of the RDF. The same situation appears in any kind of data warehousing operation, but until we tackle it properly, we have to assume that almost any data we find on the Web is stale.

At a higher level, we need to get provenance into the general consciousness. The evils of copy-paste need not be repeated here; but imagine a situation in which, whenever you did a copy operation, you automatically were given provenance data, and whenever you did a paste, either the program you were using (e.g. a text editor) knew what to do with the information or you had consciously

[2] Updated to a 2013 estimate of 31,108,077.

to throw the provenance information away. Even in writing this very short paper, I performed tens of copy-paste operations, and even here the benefits of keeping provenance data are self-evident; but the process of keeping provenance (in footnotes and citations for example) is arbitrary, and the editors and text formatters that I used gave little help with the process.

So perhaps the practical moral of these stories is that we should worry less about what provenance is and concentrate more on what we can do with it once we have it.

References

1. Amsterdamer, Y., Deutch, D., Tannen, V.: Provenance for aggregate queries. CoRR, abs/1101.1110 (2011)
2. `http://www.bbc.co.uk/news/magazine-22223190`
3. Bizer, C.: World factbook, fu berlin (UTC) (retrieved 16:30, May 4, 2013)
4. Bowers, S., McPhillips, T.M., Ludäscher, B.: Provenance in collection-oriented scientific workflows. Concurrency and Computation: Practice and Experience 20(5), 519–529 (2008)
5. Bowers, S., McPhillips, T., Ludäscher, B., Cohen, S., Davidson, S.B.: A model for user-oriented data provenance in pipelined scientific workflows. In: Moreau, L., Foster, I. (eds.) IPAW 2006. LNCS, vol. 4145, pp. 133–147. Springer, Heidelberg (2006)
6. Braun, U., Shinnar, A., Seltzer, M.I.: Securing provenance. In: HotSec (2008)
7. Buneman, P., Cheney, J., Vansummeren, S.: On the expressiveness of implicit provenance in query and update languages. ACM Trans. Database Syst. 33(4) (2008)
8. Buneman, P., Khanna, S., Tajima, K., Tan, W.C.: Archiving scientific data. ACM Trans. Database Syst. 29, 2–42 (2004)
9. Buneman, P., Khanna, S., Tan, W.-C.: Why and where: A characterization of data provenance. In: Van den Bussche, J., Vianu, V. (eds.) ICDT 2001. LNCS, vol. 1973, pp. 316–330. Springer, Heidelberg (2000)
10. Central Intelligence Agency. The World Factbook, `https://www.cia.gov/library/publications/the-world-factbook/`
11. Cheney, J., Ahmed, A., Acar, U.A.: Provenance as dependency analysis. Mathematical Structures in Computer Science 21(6), 1301–1337 (2011)
12. Cui, Y., Widom, J.: Practical lineage tracing in data warehouses. In: ICDE, pp. 367–378 (2000)
13. Davidson, S.B., Freire, J.: Provenance and scientific workflows: challenges and opportunities. In: SIGMOD Conference, pp. 1345–1350 (2008)
14. Davidson, S.B., Khanna, S., Roy, S., Stoyanovich, J., Tannen, V., Chen, Y.: On provenance and privacy. In: ICDT, pp. 3–10 (2011)
15. Deutch, D., Ives, Z., Milo, T., Tannen, V.: Caravan: Provisioning for what-if analysis. In: CIDR (2013)
16. Freire, J., Silva, C.T.: Making computations and publications reproducible with vistrails. Computing in Science and Engineering 14(4), 18–25 (2012)
17. Gil, Y., Miles, S.: Prov model primer (2013), `http://www.w3.org/TR/2013/NOTE-prov-primer-20130430/`
18. Green, T.J., Karvounarakis, G., Ives, Z.G., Tannen, V.: Provenance in orchestra. IEEE Data Eng. Bull. 33(3), 9–16 (2010)

19. Green, T.J., Karvounarakis, G., Tannen, V.: Provenance semirings. In: PODS, pp. 31–40 (2007)
20. Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers (2011)
21. Karvounarakis, G., Ives, Z.G., Tannen, V.: Querying data provenance. In: SIGMOD Conference, pp. 951–962 (2010)
22. Marinho, A., Murta, L., Werner, C., Braganholo, V., Cruz, S., Ogasawara, E., Mattoso, M.: Provmanager: a provenance management system for scientific workflows. Concurr. Comput.: Pract. Exper. 24(13), 1513–1530 (2012)
23. Moreau, L., Freire, J., Futrelle, J., McGrath, R.E., Myers, J., Paulson, P.: The open provenance model: An overview. In: Freire, J., Koop, D., Moreau, L. (eds.) IPAW 2008. LNCS, vol. 5272, pp. 323–326. Springer, Heidelberg (2008)
24. Muniswamy-Reddy, K.-K., Braun, U., David, P.M., Holland, A., Maclean, D., Margo, D., Seltzer, M., Smogor, R.: Layering in Provenance Systems. In: 2009 USENIX Annual Technical Conference, San Diego, CA (June 2009)
25. Nowakowski, P., Ciepiela, E., Harezlak, D., Kocot, J., Kasztelnik, M., Bartynski, T., Meizner, J., Dyk, G., Malawski, M.: The collage authoring environment. Procedia CS 4, 608–617 (2011)
26. Seltzer, M.: World domination through provenance (tapp 2013 keynote) (2013), https://www.usenix.org/conference/tapp13/world-domination-through-provenance
27. Sharman, J.L., Benson, H.E., Pawson, A.J., Lukito, V., Mpamhanga, C.P., Bombail, V., Davenport, A.P., Peters, J.A., Spedding, M., Harmar, A.J.: Nc-Iuphar. Iuphar-db: updated database content and new features. Nucleic Acids Research 41(Database-Issue), 1083–1088 (2013)
28. Wang, Y.R., Madnick, S.E.: A polygen model for heterogeneous database systems: The source tagging perspective. In: VLDB, pp. 519–538 (1990)
29. Woodruff, A., Stonebraker, M.: Supporting fine-grained data lineage in a database visualization environment. In: ICDE, pp. 91–102 (1997)
30. Zhao, J., Goble, C., Stevens, R., Turi, D.: Mining taverna's semantic web of provenance. Concurrency and Computation: Practice and Experience 20(5), 463–472 (2008)

A Tutorial on Trained Systems: A New Generation of Data Management Systems?

Christopher Ré

University of Wisconsin-Madison

Abstract. A new generation of data processing systems, including web search, Google's Knowledge Graph, IBM's DeepQA, and several different recommendation systems, combine rich databases with software driven by machine learning. The spectacular successes of these trained systems have been among the most notable in all of computing and have generated excitement in health care, finance, energy, and general business. But building them can be challenging even for computer scientists with PhD-level training. This tutorial will describe some of the recent progress on trained systems from both industrial and academic systems. It will also contain a walkthrough of examples of trained systems that are in daily use by scientists in Geoscience, PaleoBiology, and English Literature.

Papers, software, virtual machines that contain installations of our software, links to applications that are discussed in this talk, and our list of collaborators are available from our project page.[1] We also have a YouTube channel.[2]

Curriculum Vitae

Christopher (Chris) Re is an assistant professor in the department of computer sciences at the University of Wisconsin-Madison. The goal of his work is to enable users and developers to build applications that more deeply understand and exploit data. Chris received his PhD from the University of Washington in Seattle under the supervision of Dan Suciu. For his PhD work in probabilistic data management, Chris received the SIGMOD 2010 Jim Gray Dissertation Award. Chris's papers have received four best-paper or best-of-conference citations, including best paper in PODS 2012, best-of-conference in PODS 2010 twice, and one best-of-conference in ICDE 2009). Chris received an NSF CAREER Award in 2011 and an Alfred P. Sloan fellowship in 2013. Software and ideas from his group have been adopted by industry including Oracle and Greenplum along with scientific collaborators in neutrino physics and geoscience.

[1] http://www.cs.wisc.edu/hazy

[2] http://www.youtube.com/user/HazyResearch?feature=

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, p. 13, 2013.

Querying Big Social Data

Wenfei Fan*

University of Edinburgh and Beihang University

Abstract. Big data poses new challenges to query answering, from computational complexity theory to query evaluation techniques. Several questions arise. What query classes can be considered tractable in the context of big data? How can we make query answering feasible on big data? What should we do about the quality of the data, the other side of big data? This paper aims to provide an overview of recent advances in tackling these questions, using social network analysis as an example.

1 Introduction

Big data refers to data that cannot be processed or analyzed using traditional processes or tools, *e.g.,* when the volume of data is "big" such as in PetaByte (PB, 10^{15} bytes) or ExaByte (EB, 10^{18} bytes). As an example, let us consider social networks, which are typically modeled as graphs. In such a graph, a node denotes a person, carrying attributes such as label, keywords, blogs, comments, rating. Its edges indicate relationships such as marriage, friendship, co-work, advise, support and recommendation. Social graphs are often "big". For example, Facebook has more than 1 billion users with 140 billion links[1].

Big data introduces challenges to query answering. As an example, consider graph pattern matching, which is commonly used in social network analysis. Given a social graph G and a pattern query Q, *graph pattern matching* is to find the set $M(Q, G)$ of all matches for Q in G, as illustrated below.

Example 1. Consider the structure of a drug trafficking organization [30], depicted as a graph pattern Q_0 in Fig. 1. In such an organization, a "boss" (B) oversees the operations through a group of assistant managers (AM). An AM supervises a hierarchy of low-level field workers (FW), up to 3 levels as indicated by the edge label 3. The FWs deliver drugs, collect cash and run other errands. They report to AMs directly or indirectly, while the AMs report directly to the boss. The boss may also convey messages through a secretary (S) to the top-level FWs as denoted by the edge label 1. A drug ring G_0 is also shown in Fig. 1 in which $A_1, \ldots, A_m$ are AMs, while A_m is both an AM and the secretary (S).

To identify all suspects in the drug ring, we want to find matches $M(Q_0, G_0)$ for Q_0 in G_0. Here graph pattern matching is traditionally defined as follows:

* Fan is spported in part by EPSRC EP/J015377/1, UK, the RSE-NSFC Joint Project Scheme, and the 973 Program 2012CB316200 and NSFC 61133002 of China.

[1] `http://www.facebook.com/press/info.php?statistics`

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 14–28, 2013.

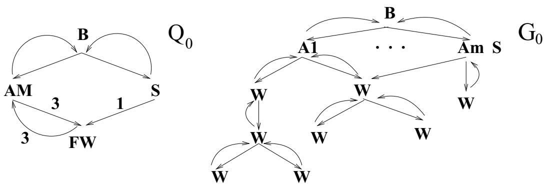


Fig. 1. Drug trafficking: Pattern and social graphs

(1) subgraph isomorphism [35]: $M(Q_0, G_0)$ is the set of all subgraphs G' of G_0 isomorphic to Q_0, *i.e.,* there exists a *bijective function* h from the nodes of Q_0 to those of G' such that (u, u') is an edge in Q_0 iff $(h(u), h(u'))$ is an edge in G'; or

(2) graph simulation [28]: $M(Q_0, G_0)$ is the maximum binary *relation* $S \subseteq V_Q \times V$, where V_Q and V are the set of nodes in Q_0 and G_0, respectively, such that

– for each node u in V_Q, there exists a node v in V such that $(u, v) \in S$, and
– for each pair $(u, v) \in S$ and each edge (u, u') in Q, there exists an edge (v, v') in G such that $(u', v') \in S$. □

No matter whether graph pattern matching is defined in terms of subgraph isomorphism or graph simulation, it involves *data-intensive* computation when graph G is "big". To develop effective algorithms for computing the set $M(Q, G)$ of matches for Q in big G, we need to answer the following questions.

(1) What query classes are tractable on big data? A class $\mathcal{Q}$ of queries is traditionally considered *tractable* if there exists an algorithm for answering its queries in time bounded by a polynomial (PTIME) in the size of the input, *e.g.,* a social graph and a pattern query [1]. That is, $\mathcal{Q}$ is considered *feasible* if its worst-case time complexity is PTIME. For graph pattern queries, it is NP-complete to determine whether there exists a match for Q in G when matching is defined with subgraph isomorphism, and it takes $O(|Q|^2 + |Q||G| + |G|)^2$ time to compute $M(Q, G)$ with graph simulation [21]. As will be seen shortly, however, PTIME or even linear-time algorithms are often beyond reach in the context of big data! This suggests that we revise the traditional notion of tractable queries, so that we can decide, given $\mathcal{Q}$, whether it is feasible to evaluate the queries of $\mathcal{Q}$ on big data.

(2) How can we make query answering feasible on big data? When a query class $\mathcal{Q}$ is not tractable on big data, we may be able to transform $\mathcal{Q}$ to an "equivalent" class $\mathcal{Q}'$ of queries that operate on smaller datasets. That is, we reduce the big data for $\mathcal{Q}$ to "small data" for $\mathcal{Q}'$, such that it is feasible to answer the queries of $\mathcal{Q}$. When querying big data, one often thinks of MapReduce [7] and Hadoop[2]. Nevertheless, MapReduce and Hadoop are not the only way to query big data. We will see that this is the case for graph pattern matching with simulation.

[2] http://hadoop.apache.org/

(3) In the context of big data it is often cost-prohibitive to compute exact answers to our queries. That is, algorithms for querying big data are often necessarily *inexact*. This is particularly evident when we want to find matches for our patterns in big social graphs based on subgraph isomorphism. Hence we may have to settle with *heuristics*, "quick and dirty" algorithms which return feasible answers. To this end, we naturally want *approximation algorithms*, *i.e.,* heuristics which find answers that are guaranteed to be not far from the exact answers [6, 36]. However, traditional approximation algorithms are mostly PTIME algorithms for NP optimization problems (NPOs). In contrast, we need approximation algorithms for answering queries on big data rather than for NPOs, even when the queries are known in PTIME, such that the algorithms are tractable on big data.

(4) When we talk about the challenges introduced by big data, we often refer to the difficulty of coping with the sheer size of the data only. Nonetheless, the quality of the data is as important and challenging as its quantity. When the quality of the data is poor, answers to our queries in the data may be *inaccurate* or even *incorrect*! Indeed, one of the dimensions of big data is its *veracity*, "as 1 in 3 business leaders don't trust the information they use to make decisions"[3]. Referring to Example 1, poor data may lead to false accusation against innocents or letting go of real drug dealers. Already challenging even for "small" relational data, data quality management is far more difficult for big data.

This paper provides an overview of recent advances in tackling these questions. We present a revision of tractable query classes in the context of big data [10] (Section 2), and a set of effective techniques beyond MapReduce for graph pattern matching with simulation [12–17, 27] (Section 3). We revisit traditional approximation algorithms for querying big data [5] (Section 4). Finally, we highlight the need for studying the quality of big data (Section 5).

2 Tractable Query Classes on Big Data

We start with an examination of query evaluation on big data, including but not limited to graph pattern matching. To develop algorithms for answering a class $\mathcal{Q}$ of queries on big data, we want to know whether $\mathcal{Q}$ is *tractable*, *i.e.,* whether its queries can be evaluated on the big data within our available resources such as time. Traditionally $\mathcal{Q}$ is considered (a) "good" (tractable) if there exists a PTIME algorithm for evaluating its queries, (b) "bad" (intractable) if it is NP-hard to decide, given a query $Q \in \mathcal{Q}$, a dataset D and an element t, whether $t \in Q(D)$, *i.e.,* t is an answer to Q in D; and (c) "ugly" if the membership problem is EXPTIME-hard. This is, however, no longer the case when it comes to big data.

Example 2. Consider a dataset D of 1PB. Assuming the fastest Solid State Drives (SSD) with disk scanning speed of 6GB/s[4], a linear scan of D will take at

[3] `http://www-01.ibm.com/software/data/bigdata/`

[4] `http://www.fastestssd.com/featured/ssd-rankings-the-fastest-solid-state-drives/#pcie`

least 166,666 seconds or 1.9 days. That is, even linear-time algorithms, a special case of PTIME algorithms, may no longer be feasible on big data! □

There has been recent work on revising the traditional computational complexity theory to characterize data-intensive computation on big data. The revisions are defined in terms of computational costs [10], communication (coordination) rounds [20, 25], or MapReduce steps [24] and data shipments [2] in the MapReduce framework [7]. Here we focus on computational costs [10].

One way to cope with big data is to separate offline and online processes. We preprocess the dataset D by, *e.g.,* building indices or compressing the data, yielding D', such that all queries of $\mathcal{Q}$ on D can be evaluated on D' *online* efficiently. When the data is mostly static or when D' can be maintained efficiently, the preprocessing step can be considered as an *offline* process with a *one-time cost.*

Example 3. Consider a class $\mathcal{Q}_1$ of selection queries. A query $Q_1 \in \mathcal{Q}_1$ on a relation D is to find whether there exists a tuple $t \in D$ such that $t[A] = c$, where A is an attribute of D and c is a constant. A naive evaluation of Q_1 would require a linear scan of D. In contrast, we can first build a B^+-tree on the values of the A column in D, in a one-time preprocessing step offline. Then we can answer *all queries* $Q_1 \in \mathcal{Q}_1$ on D in $O(\mathsf{log}|D|)$ time using the indices. That is, we no longer need to scan D when processing *each* query in $\mathcal{Q}_1$. When D consists of 1PB of data, we can get the results in 5 seconds with the indices rather than 1.9 days. □

The idea has been practiced by database people for decades. Following this, below we propose a revision of the traditional notion of tractable query classes.

To be consistent with the complexity classes of decision problems, we consider Boolean queries, such as Boolean selection queries given in Example 3. We represent a class $\mathcal{Q}$ of Boolean queries as a *language* S *of pairs* $\langle D, Q\rangle$, where Q is a query in $\mathcal{Q}$, D is a database on which Q is defined, and $Q(D)$ is true. In other words, S can be considered as a binary relation such that $\langle D, Q\rangle \in S$ if and only if $Q(D)$ is true. We refer to S as *the language for* $\mathcal{Q}$.

We say that a language S of pairs is *in complexity class* $\mathsf{C_Q}$ if it is in $\mathsf{C_Q}$ to decide whether a pair $\langle D, Q\rangle \in S$. Here $\mathsf{C_Q}$ may be the sequential complexity class P or the parallel complexity class NC, among other things. The complexity class P consists of all decision problems that can be solved by a deterministic Turing machine in PTIME. The parallel complexity class NC, *a.k.a.* Nick's Class, consists of all decision problems that can be solved by taking polynomial time in the logarithm of the problem size (parallel polylog-time) on a PRAM (parallel random access machine) with polynomially many processors (see, *e.g.,* [18, 22]).

Π-Tractable Queries. Consider complexity classes $\mathsf{C_P}$ and $\mathsf{C_Q}$. We say that a class $\mathcal{Q}$ of queries is in $(\mathsf{C_P}, \mathsf{C_Q})$ if there exist a $\mathsf{C_P}$-computable preprocessing function Π and a language S' of pairs such that for all datasets D and queries $Q \in \mathcal{Q}$,

- $\langle D, Q\rangle$ is in the language S of pairs for $\mathcal{Q}$ if and only if $\langle \Pi(D), Q\rangle \in S'$, and
- S' is in $\mathsf{C_Q}$, *i.e.,* the language S' of pairs $\langle \Pi(D), Q\rangle$ is in $\mathsf{C_Q}$.

Intuitively, function $\Pi(\cdot)$ preprocesses D and generates another structure $D' = \Pi(D)$ offline, in $\mathsf{C_P}$. After this, for *all queries* $Q \in \mathcal{Q}$ that are defined on D, $Q(D)$ can be answered by evaluating $Q(D')$ online, in $\mathsf{C_Q}$. Here $\mathsf{C_P}$ indicates the cost we can afford for preprocessing, and $\mathsf{C_Q}$ the cost of online query processing. Depending on $D' = \Pi(D)$, we may let $\mathsf{C_Q}$ be P if D' is sufficiently small such that PTIME evaluation of $Q(D')$ is feasible, *i.e.,* if $\Pi(D)$ reduces big data D to "small data" D'. Otherwise we may choose NC for $\mathsf{C_Q}$, in parallel polylog-time.

We use $\mathsf{\Pi T^0_Q}$ to denote the set of all $(\mathsf{P}, \mathsf{NC})$ query classes, referred to as the set of $\mathsf{\Pi}$*-tractable* query classes, *i.e.,* when $\mathsf{C_P}$ is P and $\mathsf{C_Q}$ is NC. We are particularly interested in $\mathsf{\Pi T^0_Q}$ for the following reasons. (a) As shown in Example 3, parallel polylog-time is feasible on big data. Moreover, NC is robust and well-understood. It is one of the few parallel complexity classes whose connections with classical sequential complexity classes have been well studied (see, *e.g.,* [18]). Further, a large class of NC algorithms can be implemented in the MapReduce framework, which is widely used in cloud computing and data centers for processing big data, such that if an NC algorithm takes t time, than its corresponding MapReduce counterpart takes $O(t)$ MapReduce rounds [24]. (b) We consider PTIME preprocessing feasible since it is a *one-time* price and is performed *off-line.* In addition, P is robust and well-studied [18]. Moreover, by requiring that $\Pi(\cdot)$ is in PTIME, the size of the preprocessed data $\Pi(D)$ is bounded by a polynomial. When PTIME is too big a price to pay, we may preprocess D with parallel processing, by allocating more resources (*e.g.,* computing nodes) to it than to online query answering. Here we simply use PTIME for $\mathsf{C_P}$ to focus on the main properties of query answering with preprocessing.

Example 4. As shown in Example 3, query class $\mathcal{Q}_1$ is in $\mathsf{\Pi T^0_Q}$. Indeed, function $\Pi(\cdot)$ preprocesses D by building B^+-trees on attributes of D in PTIME. After this, for any (A, c) denoting a query in $\mathcal{Q}_1$, whether there exists $t \in D$ such that $t[A] = c$ can be decided in $O(\log|D|)$ time by using the indices $\Pi(D)$. □

Making Query Classes Π-Tractable. Many query classes $\mathcal{Q}$ are not Π-tractable. For instance, unless $\mathsf{P} = \mathsf{NC}$, we are not aware of any NC algorithm for graph pattern matching even when matching is defined in terms of graph simulation. Nonetheless, some $\mathcal{Q}$ that is not in $\mathsf{\Pi T^0_Q}$ can actually be transformed to a Π-tractable query class by means of *re-factorizations*, which re-partition the data and query parts of $\mathcal{Q}$ and identify a data set for preprocessing, such that after the preprocessing, its queries can be subsequently answered in parallel polylog-time.

More specifically, we say that a class $\mathcal{Q}$ of queries *can be made* Π*-tractable* if there exist three NC computable functions $\pi_1(\cdot)$, $\pi_2(\cdot)$ and $\rho(\cdot,\cdot)$ such that for all $\langle D, Q\rangle$ in the language S of pairs for $\mathcal{Q}$,

- $D' = \pi_1(D, Q)$, $Q' = \pi_2(D, Q)$, and $\langle D, Q \rangle = \rho(D', Q')$, and
- the query class $\mathcal{Q}' = \{Q' \mid Q' = \pi_2(D, Q), \langle D, Q \rangle \in S\}$ is Π-tractable.

Intuitively, $\pi_1(\cdot)$ and $\pi_2(\cdot)$ re-partition $x = \langle D, Q \rangle$ into a "data" part $D' = \pi_1(x)$ and a "query" part $Q' = \pi_2(x)$, and ρ is an inverse function that restores the original instance x from $\pi_1(x)$ and $\pi_2(x)$. Then the data part D' can be preprocessed such that the queries $Q' \in \mathcal{Q}'$ can then be answered in parallel polylog-time. We denote by $\Pi\mathsf{T}_\mathsf{Q}$ the set of all query classes that can be *made* Π-tractable.

A form of NC-reductions $\leqslant_{fa}^{\mathsf{NC}}$ is defined for $\Pi\mathsf{T}_\mathsf{Q}$, which is transitive (*i.e.,* if $\mathcal{Q}_1 \leqslant_{fa}^{\mathsf{NC}} \mathcal{Q}_2$ and $\mathcal{Q}_2 \leqslant_{fa}^{\mathsf{NC}} \mathcal{Q}_3$ then $\mathcal{Q}_1 \leqslant_{fa}^{\mathsf{NC}} \mathcal{Q}_3$) and compatible with $\Pi\mathsf{T}_\mathsf{Q}$ (*i.e.,* if $\mathcal{Q}_1 \leqslant_{fa}^{\mathsf{NC}} \mathcal{Q}_2$ and $\mathcal{Q}_2$ is in $\Pi\mathsf{T}_\mathsf{Q}$, then so is $\mathcal{Q}_1$). The following results are known [10]:

- $\mathsf{NC} \subseteq \Pi\mathsf{T}_\mathsf{Q}^0 \subseteq \mathsf{P}$.
- Unless $\mathsf{P} = \mathsf{NC}$, $\Pi\mathsf{T}_\mathsf{Q}^0 \subset \mathsf{P}$, *i.e.,* not all PTIME queries are Π-tractable.
- There exists a *complete query class* $\mathcal{Q}$ for $\Pi\mathsf{T}_\mathsf{Q}$ under $\leqslant_{fa}^{\mathsf{NC}}$ reductions, *i.e.,* $\mathcal{Q}$ is in $\Pi\mathsf{T}_\mathsf{Q}$ and moreover, for all query classes $\mathcal{Q}' \in \Pi\mathsf{T}_\mathsf{Q}$, $\mathcal{Q}' \leqslant_{fa}^{\mathsf{NC}} \mathcal{Q}$.
- All query classes in P can be made Π-tractable by transforming them to a query class in $\Pi\mathsf{T}_\mathsf{Q}$ via $\leqslant_{fa}^{\mathsf{NC}}$ reductions.

3 Graph Pattern Matching in Big Social Data

We now study how to compute matches $M(Q, G)$ for a pattern Q in a big social graph G. We focus on matching defined in terms of graph simulation in this section, which is widely used in social data analysis such as detecting social communities and positions [4, 33]. As remarked earlier, it takes $O(|Q|^2 + |Q||G| + |G|)^2$ time to compute $M(Q, G)$ [21], a prohibitive cost when G is big. Nonetheless, we can leverage a variety of techniques commonly used by database people to reduce G to G' of smaller size via preprocessing, such that $M(Q, G')$ can subsequently be computed effectively for all patterns Q. Combinations of these techniques outperform direct implementation of simulation algorithms in MapReduce.

We first introduce a revision of graph simulation [28] for social data analysis (Section 3.1). We then present a set of matching techniques (Sections 3.2–3.6).

3.1 Bounded Simulation: Graph Simulation Revisited

Recall Example 1: we want to identify suspects involved in a drug ring by computing matches $M(Q_0, G_0)$ for pattern Q_0 in graph G_0. However, observe the following. (1) Nodes `AM` and `S` in Q_0 should be mapped to the *same* node A_m in G_0, which is not allowed by a bijection. (2) The node `AM` in Q_0 corresponds to *multiple* nodes $A_1, \ldots, A_m$ in G_0. This relationship cannot be captured by a function from the nodes of Q_0 to the nodes of G_0. (3) The edge from `AM` to `FW` in Q_0 indicates that an `AM` supervises `FW`s within 3 hops. It should be mapped to a *path* of a bounded length in G_0 rather than to an *edge*. Hence, neither subgraph isomorphism (for (1)–(3)) nor graph simulation (for (3)) is capable of identifying the drug ring G_0 as a match of Q_0. These call for revisions of the notion of graph pattern matching to accurately identify sensible matches in real-life social graphs.

To cope with this, a revision of graph simulation is proposed in [12], referred to as *bounded simulation.* To present this, we start with some notations.

Graphs and Patterns. A *data graph* is a directed graph $G = (V, E, f_A)$, where (a) V is a finite set of nodes; (b) $E \subseteq V \times V$, in which (v, v') denotes an edge from v to v'; and (c) $f_A(\cdot)$ is a function that associates each v in V with a tuple $f_A(v) = (A_1 = a_1, \ldots, A_n = a_n)$, where a_i is a constant, and A_i is referred to as an *attribute* of v, written as $v.A_i$, carrying, *e.g.,* label, keywords, blogs, rating.

A *pattern query* is defined as $Q = (V_Q, E_Q, f_v, f_e)$, where (a) V_Q is a finite set of nodes and E_Q is a set of directed edges, as defined for data graphs; (b) $f_v(\cdot)$ is a function defined on V_Q such that for each node u, $f_v(u)$ is the *predicate* of u, defined as a conjunction of atomic formulas of the form $A \ \mathsf{op}\ a$; here A denotes an attribute, a is a constant, and op is one of the comparison operators $<, \leq, =, \neq, >, \geq$; and (c) $f_e(\cdot)$ is a function defined on E_Q such that for each edge (u, u') in E_Q, $f_e(u, u')$ is either a positive integer k or a symbol $*$.

Intuitively, the predicate $f_v(u)$ of a node u specifies a *search condition.* We say that a node v in a data graph G *satisfies* the search condition of a pattern node u in Q, denoted as $v \sim u$, if for each atomic formula '$A \ \mathsf{op}\ a$' in $f_v(u)$, there exists an attribute A in $f_A(v)$ such that $v.A \ \mathsf{op}\ a$. We will allow an edge (u, u') in Q to be mapped to a path ρ in a data graph G, and $f_e(u, u')$ imposes a bound on the length of ρ. An example data graph (resp. pattern) is G_0 (resp. Q_0) of Fig. 1.

Bounded Simulation. We now present bounded simulation. A data graph G *matches* a pattern Q via *bounded simulation*, denoted by $Q \unlhd^{\mathsf{B}}_{\mathsf{sim}} G$, if there exists a binary relation $S \subseteq V_Q \times V$, referred to as a *match* in G for Q, such that

- for each node $u \in V_Q$, there exists a node $v \in V$ such that $(u, v) \in S$;
- for each pair $(u, v) \in S$, (a) $v \sim u$, and (b) for each edge (u, u') in E_Q, there exists a *path* ρ from v to v' in G such that $(u', v') \in S$, $\mathsf{len}(\rho) > 0$ and moreover, $\mathsf{len}(\rho) \leq k$ if $f_e(u, u') = k$. Here $\mathsf{len}(\rho)$ is the number of edges on ρ.

Intuitively, $(u, v) \in S$ if (1) node v in G satisfies the search condition specified by $f_v(u)$ in Q; and (2) each edge (u, u') in Q is mapped to a *path* ρ from v to v' in G ($\mathsf{len}(\rho) > 0$), such that v, v' match u, u', respectively; and moreover, when $f_e(u, u')$ is k, it indicates a bound on the length of ρ, *i.e.,* v is connected to v' within k hops. When it is $*$, ρ can be a path of an arbitrary length greater than 0.

For pattern Q_0 and graph G_0 given in Fig. 1, $Q_0 \unlhd^{\mathsf{B}}_{\mathsf{sim}} G_0$: a match S_0 in G_0 for Q_0 maps `B` to `B`, `AM` to $A_1, \ldots, A_m$, `S` to A_m, and `FW` to all the `W` nodes.

As experimentally verified in [12], bounded simulation is able to accurately identify a number of communities in real-life social networks that its traditional counterparts fail to catch. In addition, the following is known.

Theorem 1 [12]: For any pattern $Q = (V_Q, E_Q, f_v, f_e)$ and graph $G = (V, E, f_A)$, (1) there exists a unique maximum match $M(Q, G)$ in G for Q, and (2) $M(Q, G)$ can be computed in $O(|V||E| + |E_Q||V|^2 + |V_Q||V|)$ time. □

As opposed to subgraph isomorphism, bounded simulation supports (a) simulation relations rather than bijective functions, (b) search conditions based on the contents of nodes, and (c) edge-to-path mappings instead of edge-to-edge. Graph simulation is a special case of bounded simulation, by only allowing simple patterns in which (a) node labels are the only attributes, and (b) all the edges are labeled with 1, *i.e.,* edge-to-edge mappings only. In contrast to the NP-hardness of subgraph isomorphism, the complexity of bounded simulation is in PTIME, comparable to that of graph simulation since in practice, $|Q| \ll |D|$.

There have also been revisions of (bounded) simulation by, *e.g.,* incorporating edge relationships [11] and imposing locality and duality on match relations [27].

3.2 Distributed Query Processing with Partial Evaluation

Although graph pattern matching with (bounded) simulation is in PTIME, when a social graph G is big, the cost of computing $M(Q, G)$ is still prohibitive. To cope with the sheer size of G, we next present a set of approaches to computing $M(Q, G)$ on big G. The key idea of these approaches is to reduce G to smaller G' via preprocessing, such that graph pattern matching in G' is feasible.

We start with distributed query processing, based on partial evaluation. Partial evaluation has proven useful in a variety of areas including compiler generation, code optimization and dataflow evaluation (see [23] for a survey). Intuitively, given a function $f(s, d)$ and part of its input s, partial evaluation is to specialize $f(s, d)$ with respect to the known input s. That is, it conducts the part of $f(s, \cdot)$'s computation that depends only on s, and generates a partial answer, *i.e.,* a residual function $f'(\cdot)$ that depends on the as yet unavailable input d.

This idea can be naturally applied to distributed graph pattern matching. Consider a pattern Q posed on a graph G that is partitioned into fragments $\mathcal{F} = (F_1, \ldots, F_n)$, where F_i is stored in site S_i. We compute $M(Q, G)$ as follows.

(1) The same query Q is posted to each fragment in $\mathcal{F}$.
(2) Upon receiving Q, each site S_i computes a *partial answer* of Q in fragment F_i, *in parallel*, by taking F_i as the known input s while treating the fragments in the other sites as yet unavailable input d.
(3) A coordinator site S_c collects partial answers from all the sites. It then assembles the partial answers and finds $M(Q, G)$ in the entire graph G.

The idea has proven effective for evaluating reachability queries defined in terms of regular expressions, which are a special case of pattern queries [15].

Theorem 2 [15]: On a fragmentation $\mathcal{F}$ of graph G, reachability queries Q can be answered (a) by visiting each site once, (b) in $O(|F_m||Q|^2 + |Q|^2|V_f|^2)$ time, and (c) with $O(|Q|^2|V_f|^2)$ communication cost, where F_m is the largest fragment in $\mathcal{F}$ and V_f is the set of nodes in G with edges to other fragments. □

That is, (1) the response time is dominated by the largest fragment in $\mathcal{F}$, *instead of* the size $|G|$ of G; (2) the total amount of data shipped is determined by the size of the query Q and how G is fragmented, *rather than* by $|G|$, and (3) the performance guarantees remain intact no matter how G is fragmented and distributed. As opposed to MapReduce [7], this approach does not require us to organize our data in $\langle \mathsf{key}, \mathsf{value}\rangle$ pairs or re-distribute the data. Moreover, it has performance guarantees on both response time and communication cost.

When G is not already partitioned and distributed, one may first partition G as preprocessing, such that the evaluation of Q in each fragment is feasible.

3.3 Query Preserving Graph Compression

Another approach to reducing the size of big graph G is by means of compressing G, relative to a class $\mathcal{Q}$ of queries of users' choice, *e.g.,* graph pattern queries. More specifically, a *query preserving graph compression* for $\mathcal{Q}$ is a pair $\langle R, P\rangle$, where $R(\cdot)$ is a *compression function*, and $P(\cdot)$ is a *post-processing function.* For any graph G, $G_c = R(G)$ is the *compressed graph* computed from G by $R(\cdot)$, such that (1) $|G_c| \leq |G|$, and (2) *for all queries* $Q \in \mathcal{Q}$, $Q(G) = P(Q(G_c))$. Here $P(Q(G_c))$ is the result of post-processing the answers $Q(G_c)$ to Q in G_c.

That is, we preprocess G by computing the compressed G_c of G offline. After this step, for any query $Q \in \mathcal{Q}$, the answers $Q(G)$ to Q in the big G can be computed by evaluating the same Q on the smaller G_c online. Moreover, $Q(G_c)$ can be computed *without decompressing* G_c. Note that the compression schema is *lossy*: we do not need to restore the original G from G_c. That is, G_c only needs to retain the information necessary for answering queries in $\mathcal{Q}$, and hence achieves *better* compression ratio than lossless compression schemes.

For a query class $\mathcal{Q}$, if G_c can be computed in PTIME and moreover, queries in $\mathcal{Q}$ can be answered using G_c in parallel polylog-time, perhaps by combining with other techniques such as indexing, then $\mathcal{Q}$ is Π-tractable.

The effectiveness of the approach has been verified in [14], for graph pattern matching with (bounded) simulation, and for reachability queries a special case.

Theorem 3 [14]: There exists a graph pattern preserving compression $\langle R, P\rangle$ for bounded simulation, such that for any graph $G = (V, E, f_A)$, $R(\cdot)$ is in $O(|E| \log |V|)$ time, and $P(\cdot)$ is in linear time in the size of the query answer. □

This compression scheme reduces the sizes of real-life social graphs by 98% and 57%, and query evaluation time by 94% and 70% on average, for reachability queries and pattern queries with (bounded) simulation, respectively. Better still, compressed G_c can be efficiently maintained. Given a graph G, a compressed graph $G_c = R(G)$ of G, and updates ΔG to G, we can compute changes ΔG_c to G_c such that $G_c \oplus \Delta G_c = R(G \oplus \Delta G)$, *without decompressing* G_c [14]. As a result, for each graph G, we need to compute its compressed graph G_c *once* for *all patterns.* When G is updated, G_c is incrementally maintained.

3.4 Graph Pattern Matching Using Views

Another technique commonly used by database people is query answering using views (see [19, 26] for surveys). Given a query $Q \in \mathcal{Q}$ and a set $\mathcal{V}$ of view definitions, *query answering using views* is to reformulate Q into another query Q' such that (a) Q and Q' are equivalent, *i.e.,* for all datasets D, Q and Q' have the same answers in D, and moreover, (b) Q' refers only to $\mathcal{V}$ and its extensions $\mathcal{V}(D)$.

View-based query answering suggests another approach to reducing big data to small data. Given a big graph G, one may identify a set $\mathcal{V}$ views (pattern queries) and materialize the set $M(\mathcal{V}, G)$ of matches for patterns of $\mathcal{V}$ in G, as a preprocessing step offline. Then matches for patterns Q can be computed online by using $M(\mathcal{V}, G)$ only, *without accessing the original big* G. In practice, $M(\mathcal{V}, G)$ is typically much smaller than G, and can be incrementally maintained and adaptively adjusted to cover various patterns. For example, for graph pattern matching with bounded simulation, $M(\mathcal{V}, G)$ is no more than 4% of the size of G on average for real-life social graphs G. Further, the following has been shown [17].

Theorem 4 [17]: Given a graph pattern Q and a set $\mathcal{V}$ of view definitions, (1) it is in $O(|Q|^2|\mathcal{V}|)$ time to decide whether Q can be answered by using $\mathcal{V}$; and if so, (2) Q can be answered in $O(|Q||M(\mathcal{V}, G)| + |M(\mathcal{V}, G)|^2)$ time. □

Contrast these with the complexity of graph pattern matching with bounded simulation. Note that $|Q|$ and $|\mathcal{V}|$ are sizes of pattern queries and are typically much smaller than G. Moreover, $|M(\mathcal{V}, G)|$ is about 4% of $|G|$ (*i.e.,* $|V| + |E|$) on average. As verified in [17], graph pattern matching using views takes no more than 6% of the time needed for computing $M(Q, G)$ directly in G on average.

3.5 Incremental Graph Pattern Matching

Incremental techniques also allow us to effectively evaluate queries on big data. Given a pattern Q and a graph G, as preprocessing we compute $M(Q, G)$ once. When G is updated by ΔG, instead of recomputing $M(Q, G \oplus \Delta G)$ starting from scratch, we incrementally compute ΔM such that $M(Q, G \oplus \Delta G) = M(Q, G) \oplus \Delta M$, to minimize unnecessary recomputation. In real life, ΔG is typically small: only 5% to 10% of nodes are updated weekly [31]. When ΔG is small, ΔM is often small as well, and is much less costly to compute than $M(Q, G \oplus \Delta G)$.

The benefit is more evident if there exists a bounded incremental matching algorithm. As argued in [32], incremental algorithms should be analyzed in terms of $|\mathsf{CHANGED}| = |\Delta G| + |\Delta M|$, the size of changes in the input and output, which represents the updating costs that are *inherent to* the incremental problem itself. An incremental algorithm is said to be *semi-bounded* if its cost can be expressed as a polynomial of $|\mathsf{CHANGED}|$ and $|Q|$ [13]. That is, its cost depends only on the size of *the changes* and the size of *pattern* Q, *independent of* the size of big graph G. A semi-bounded incremental algorithm often reduces big graph G to small data, since Q and $|\mathsf{CHANGED}|$ are typically small in practice.

Theorem 5 [13]: There exists a semi-bounded incremental algorithm, in $O(|\Delta G|(|Q||\mathsf{CHANGED}| + |\mathsf{CHANGED}|^2))$ time, for graph pattern matching defined in terms of bounded simulation. □

In general, a query class $\mathcal{Q}$ can be considered Π-tractable if (a) preprocessing $Q(D)$ is in PTIME, and (b) $Q(D \oplus \Delta D)$ can be incrementally computed in parallel polylog-time. If so, it is feasible to answer Q in response to changes to big data D.

3.6 Top-k Graph Pattern Matching

In social data analysis we often want to find matches of a designated pattern node u_o in Q as "query focus" [3]. That is, we just want those nodes in a social graph G that are matches of u_o in $M(Q,G)$, rather than the entire set $M(Q,G)$ of matches for Q. Indeed, a recent survey shows that 15% of social queries are to find matches of specific pattern nodes [29]. This is how graph search[5] of Facebook is conducted on its social graph. Moreover, it often suffices to find top-k matches of u_o in $M(Q,G)$. More specifically, assume a scoring function $s(\cdot)$ that given a match v of u_o, returns a non-negative real number $s(v)$. For a positive integer k, *top-k graph pattern matching* is to find a set U of matches of u_o in $M(Q,G)$, such that U has exactly k matches and moreover, for any k-element set U' of matches of u_o, $s(U') \leq s(U)$, where $s(U)$ is defined as $\Sigma_{v \in U} s(v)$. When there exist less than k matches of u_o in $M(Q,G)$, U includes all such matches.

This suggests that we develop algorithms to find top-k matches with *the early termination property* [8], *i.e.,* they stop as soon as a set of top-k matches is found, *without* computing the entire $M(Q,G)$. While the worst-case time complexity of such algorithms may be no better than their counterparts for computing the entire $M(Q,G)$, they may only need to inspect part of big G, without paying the price of full-fledged graph pattern matching. Indeed, for graph pattern matching with simulation on real-life social graphs, it has been shown that top-k matching algorithms just inspect 65%–70% of the matches in $M(Q,G)$ on average [16].

4 Approximation Algorithms for Querying Big Data

Strategies such as those given above help us make the evaluation of *some* queries tractable on big data. However, it is still beyond reach to find exact answers to many queries in big data. An example is graph pattern matching defined in terms of subgraph isomorphism: it is NP-complete to decide whether there exists a match. As remarked earlier, even for queries that can be answered in PTIME, it is often too costly and infeasible to compute their exact answers in the context of big data. As a result, we have to evaluate these queries by using *inexact* algorithms, preferably approximation algorithms with performance guarantees.

Previous work on this topic has mostly focused on developing PTIME approximation algorithms for NP-optimization problems (NPOs) [6, 18, 36]. An NPO A has a set I of instances, and for each instance $x \in I$ and each feasible solution

[5] `http://www.facebook.com/about/graphsearch`

y of x, there exists a positive score $m(x, y)$ indicating the quality measure of y. Consider a function $r(\cdot)$ from natural numbers to $(1, \infty)$. An algorithm T is called *a r-approximation algorithm for problem A* if for each instance $x \in I$, T computes a feasible solution y of x such that $R(x, y) \leq r(|x|)$, where $R(x, y)$ is the *performance ratio* of y *w.r.t.* x, defined as follows [6]:

$$R(x,y) = \begin{cases} \mathsf{opt}(x)/m(x,y) & \text{when } A \text{ is a maximization problem} \\ m(x,y)/\mathsf{opt}(x) & \text{when } A \text{ is a minimization problem} \end{cases}$$

where $\mathsf{opt}(x)$ is the optimal solution of x. That is, while the solution y found by $T(x)$ may not be optimal, it is not too far from $\mathsf{opt}(x)$ (*i.e.,* bounded by $r(|x|)$).

However, PTIME approximation algorithms that directly operate on the original instances of a problem may not work well when querying big data.

(1) As shown in Example 2, PTIME algorithms on x may be beyond reach in practice when x is big. Moreover, approximation algorithms are needed for problems that are traditionally considered tractable [18], not limited to NPO.

(2) In contrast to NPOs that ask for a single optimum, query evaluation is to find *a set* of query answers in a dataset. Thus we need to revise the notion of performance ratios to evaluate the quality of a set of feasible answers.

After the topic has been studied for decades, it is unlikely that we can expect soon to have a set of algorithms that on one hand, have low enough complexity to be tractable on big data, and on the other hand, have a nice performance ratio.

Data-driven Approximation. To cope with this, we propose to develop algorithms that work on data with "resolution" lower than the original instances, and strike a *balance* between the efficiency (scalability) and the performance ratio [5]. Consider a pair $\langle D, Q\rangle$ that represents an instance x, where Q is a query and D is a dataset (see Section 2). When D is big, we reduce D to D' of manageable size, and develop algorithms that are feasible when operating on D'.

More specifically, consider a function $\alpha(\cdot)$ that takes $|D|$ as input, and returns a number in $(0, 1]$. We use a *transformation function* $f(\cdot)$ that given D, reduces D to $D' = f(D)$ with *resolution* $\alpha(|D|)$ such that $|D'| \leq \alpha(|D|) \cdot |D|$. We also use a *query rewriting function* $F : \mathcal{Q} \to \mathcal{Q}$ for a query class $\mathcal{Q}$ that, given any $Q \in \mathcal{Q}$, returns another query $F(Q)$ in $\mathcal{Q}$. Based on these, we introduce the following.

An algorithm T is called a (α, r)-*approximation algorithm* for $\mathcal{Q}$ if there exist functions $f(\cdot)$ and $F(\cdot)$ such that for any dataset D,

(1) $D' = f(D)$ and $|D'| \leq \alpha(|D|)|D|$; and

(2) for each query Q in $\mathcal{Q}$ defined on D, $Q' = F(Q)$, and algorithm T computes $Y = Q'(D')$ such that the performance ratio $R(\langle D, Q\rangle, Y) \leq r(|D|)$.

Intuitively, $f(\cdot)$ is an *offline* process that reduces big data D to small D' with *a lower resolution* $\alpha(|D|)$. After this, *for all* queries Q in $\mathcal{Q}$ posed on D, T is used to evaluate $Q' = F(Q)$ in D' as an *online* process, such that the feasible answers $Y = Q'(D')$ computed by T in D' are not too far from the *exact answers* $Q(D)$ in D. To evaluate the accuracy of Y, we need to extend the notion of performance ratio $R(\cdot, \cdot)$ to measure how close *a set* of feasible query answers Y is to the set $Q(D)$ of *exact* answers. There are a variety of choices for defining

$R(\cdot,\cdot)$, depending on the application domain in which T is developed (see [5] for details).

Example 5. A weighted undirected graph is defined as $G = (V, E, w)$, where for each edge e in E, $w(e)$ is the weight of e. Given G and two nodes s, t in V, we want to compute the distance $\mathsf{dist}(s,t)$ between s and t in G, *i.e.,* the minimum sum of the weights of the edges on a path from s to t for all such paths in G.

There exist exact algorithms for computing $\mathsf{dist}(s,t)$ in $O(|E|)$-time (cf. [34]). However, when G is big, we need more efficient algorithms. It has been shown in [34] that for any constant $k \geq 1$, one can produce a data structure of size $O(k|V|^{1+1/k})$. After this offline process, all distance queries on G can be answered in $O(k)$ time (constant time) online by using the structure, with a constant performance ratio $2k-1$ [34]. That is, there exists a (α, r)-approximation algorithm for distance queries, with $\alpha(|G|) = |V|^{1+1/k}/(|V| + |E|)$ and $r = 2k - 1$. □

Data-driven approximation aim to explores the connection between *the resolution* of data and *the performance ratio* of algorithms, and speed up the *online* process. As remarked earlier, the choice of $f(\cdot)$ and T depends on what cost we can afford for offline preprocessing and what algorithms are tractable on big data. When $\alpha(|D|)$ is sufficiently small (*e.g.,* below a certain threshold ξ), $f(D)$ reduces "big data" D to "small data" D', on which a PTIME algorithm T is feasible. However, if D' remains "big", *i.e.,* when $\alpha(|D|) \geq \xi$, we may require T to be in NC. To cope with big D, the offline preprocessing step may require more resources such as computing nodes for parallel processing than online query evaluation.

5 Data Quality: The Other Side of Big Data

The study of querying big (social) data is still in its infancy. There is much more to be done. In particular, a complexity class that captures queries tractable on big data has to be identified, to characterize both computational and communication costs. Complete problems and reductions for the complexity class should be in place, so that we can effectively decide whether a class of queries is tractable on big data and whether a query class can be reduced to another one that we know how to process. In addition, more effective techniques for querying big data should be developed so that (combinations of) the techniques can improve query processing by MapReduce. Furthermore, the connection between data resolution and performance ratio needs a full treatment. Given a resolution, we should be able to determine what performance ratio we can expect, and vice versa.

We have so far focused on how to cope with the volume (quantity) of big data. Nonetheless, *data quality* is as important as data quantity. As an example, consider tuples below that represent suspects for a secretary S in a drug ring, identified by graph pattern matching in social networks (recall Q_0 and G_0 of Fig. 1):

	FN	LN	AC	street	city	state	zip	status
t_1:	Mary	Smith	212	Mtn Ave	MH	NJ	10007	single
t_2:	Mary	Smith	908	Mtn Ave	MH	NJ	07974	single
t_3:	Mary	Luth	212	Broadway	NY	NY	10007	married

Each tuple in the table specifies a suspect: her name (FN and LN), area code AC, address (street, city, state, zip code), and marital status, extracted from social networks. Consider the following simple queries about the suspects.

(1) Query Q_1 is to find how many suspects are based in New Jersey. By counting those tuples t with $t[\mathsf{state}]$ = "NJ", we get 2 as its answer. However, the answer may be *incorrect*. Indeed, (a) the data in tuple t_1 is *inconsistent*: $t_1[\mathsf{AC}] = 212$ is an area code for New York, and it has conflict with $t_1[\mathsf{state}]$ (NJ). Hence NJ may not be the true value of $t_1[\mathsf{state}]$. (b) The data in the table may be *incomplete*. That is, some suspects may not use social networks and hence, are overlooked. (c) Tuples t_1, t_2 and t_3 may refer to the same person and hence, may be *duplicates*. In light of these data quality issues, we cannot trust the answer to query Q_1.

(2) Suppose that the table above is complete, t_1, t_2 and t_3 refer to the same person Mary, and all their attribute values were once the true values of Mary but some may have become obsolete. Now query Q_2 is to find Mary's current last name. We do not know whether it is Smith or Luth. However, we know that marital status can only change from single to married, and that her last name and marital status are correlated. From these we can conclude that the answer to Q_2 is Luth.

This example tells us the following. First, when the quality of the data is poor, we cannot trust answers to our queries no matter how big data we can handle and how efficient we can process our queries. Second, data quality analyses help us improve the quality of our query answers. However, already difficult for (small) relational data (see [9] for a survey), the study of data consistency, accuracy, currency, deduplication and information completeness is far more challenging for big data. Indeed, big data is typically heterogeneous (variety), time-sensitive (velocity), of low-quality (veracity) and big (volume). Despite these, data quality is a must for us to study if we want to make practical use of big data.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Afrati, F.N., Ullman, J.D.: Optimizing joins in a map-reduce environment. In: EDBT (2010)
3. Bendersky, M., Metzler, D., Croft, W.: Learning concept importance using a weighted dependence model. In: WSDM (2010)
4. Brynielsson, J., Högberg, J., Kaati, L., Martenson, C., Svenson, P.: Detecting social positions using simulation. In: ASONAM (2010)
5. Buneman, P., Fan, W.: Data driven approximation algorithms for querying big data (2013) (unpublished manuscript)
6. Crescenzi, P., Kann, V., Halldórsson, M.: A compendium of NP optimization problems, `http://www.nada.kth.se/~viggo/wwwcompendium/`
7. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM 51(1) (2008)
8. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. JCSS 66(4), 614–656 (2003)

9. Fan, W., Geerts, F.: Foundations of Data Quality Management. Morgan & Claypool Publishers (2012)
10. Fan, W., Geerts, F., Neven, F.: Making queries tractable on big data with preprocessing. In: PVLDB (2013)
11. Fan, W., Li, J., Ma, S., Tang, N., Wu, Y.: Adding regular expressions to graph reachability and pattern queries. In: ICDE (2011)
12. Fan, W., Li, J., Ma, S., Tang, N., Wu, Y., Wu, Y.: Graph pattern matching: From intractability to polynomial time. In: PVLDB (2010)
13. Fan, W., Li, J., Tan, Z., Wang, X., Wu, Y.: Incremental graph pattern matching. In: SIGMOD (2011)
14. Fan, W., Li, J., Wang, X., Wu, Y.: Query preserving graph compression. In: SIGMOD (2012)
15. Fan, W., Wang, X., Wu, Y.: Performance guarantees for distributed reachability queries. In: PVLDB (2012)
16. Fan, W., Wang, X., Wu, Y.: Diversified top-k graph pattern matching (2013) (unpublished manuscript)
17. Fan, W., Wang, X., Wu, Y.: Graph pattern matching using views (2013) (unpublished manuscript)
18. Greenlaw, R., Hoover, H.J., Ruzzo, W.L.: Limits to Parallel Computation: P-Completeness Theory. Oxford University Press (1995)
19. Halevy, A.Y.: Answering queries using views: A survey. VLDB J. 10(4) (2001)
20. Hellerstein, J.M.: The declarative imperative: Experiences and conjectures in distributed logic. SIGMOD Record 39(1), 5–19 (2010)
21. Henzinger, M.R., Henzinger, T., Kopke, P.: Computing simulations on finite and infinite graphs. In: FOCS (1995)
22. Johnson, D.S.: A catalog of complexity classes. In: Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A), The MIT Press (1990)
23. Jones, N.D.: An introduction to partial evaluation. ACM Comput. Surv. 28(3), 480–503 (1996)
24. Karloff, H.J., Suri, S., Vassilvitskii, S.: A model of computation for MapReduce. In: SODA (2010)
25. Koutris, P., Suciu, D.: Parallel evaluation of conjunctive queries. In: PODS (2011)
26. Lenzerini, M.: Data integration: A theoretical perspective. In: PODS (2002)
27. Ma, S., Cao, Y., Fan, W., Huai, J., Wo, T.: Capturing topology in graph pattern matching. PVLDB 5(4) (2011)
28. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
29. Morris, M., Teevan, J., Panovich, K.: What do people ask their social networks, and why? A survey study of status message Q&A behavior. In: CHI (2010)
30. Natarajan, M.: Understanding the structure of a drug trafficking organization: a conversational analysis. Crime Prevention Studies 11, 273–298 (2000)
31. Ntoulas, A., Cho, J., Olston, C.: What's new on the Web? The evolution of the Web from a search engine perspective. In: WWW (2004)
32. Ramalingam, G., Reps, T.: On the computational complexity of dynamic graph problems. TCS 158(1-2) (1996)
33. Terveen, L., McDonald, D.W.: Social matching: A framework and research agenda. ACM Trans. Comput.-Hum. Interact. 12(3) (2005)
34. Thorup, M., Zwick, U.: Approximate distance oracles. JACM 52(1), 1–24 (2005)
35. Ullmann, J.R.: An algorithm for subgraph isomorphism. JACM 23(1), 31–42 (1976)
36. Vazirani, V.V.: Approximation Algorithms. Springer (2003)

Database Research Challenges and Opportunities of Big Graph Data

Alexandra Poulovassilis

London Knowledge Lab, Birkbeck, University of London, UK
ap@dcs.bbk.ac.uk

1 Overview

Large volumes of graph-structured data are becoming increasingly prevalent in areas such as

– social and professional network analysis
– recommendation services, such as product advertisement, news and media alerts, learning resource recommendation, itinerary recommendation
– scientific computing: life and health sciences, physical sciences
– crime investigation and intelligence gathering
– telecoms network management, for dependency analysis, root cause analysis, location-based service provision
– linked open data
– geospatial data
– business process management: logistics, finance chains, fraud detection, risk analysis, asset management
– organization management

Graph-structured data differs from other “big” data in its greater focus on the relationships between entities, regarding these relationships as important as the entities themselves, and allowing the possibility of modelling the attributes of relationships just as for entities, specifying constraints on relationships, and undertaking querying, analysis and reasoning over relationships.

Processing that may be undertaken with graph data includes on-line transaction processing and querying of the data on the one hand [23], and more computationally- and data-intensive off-line search, mining and analytics on the other [10].

Graph-oriented data processing algorithms may be applied to data that is stored in conventional databases, in NoSQL databases, or in specialised triple stores or graph databases. There are potentially several advantages in storing graph-structured data in specialised databases in order to undertake graph-oriented data processing: more natural support for graph data modelling [8], support for graph-oriented query languages that are better suited to formulating and optimising graph queries [25,13,2,4]; graph-specific storage and indexing for fast link and path traversal (e.g. as in Neo4J [7], DEX [18], GRAIL [26], SCARAB [14], SAINT-DB [21]), and in-database support for core graph algorithms such as subgraph matching, breadth-first/depth-first search, path finding, shortest path.

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 29–32, 2013.

Beyond centralised architectures, in the area of distributed graph data processing the aim is to handle larger volumes of graph data than can be handled on a single server, with the goal of achieving horizontal scalability. Approaches include systems that provide distributed graph processing over MapReduce-based frameworks, such as Pegasus [15]; BSP (Bulk Synchronous Processing) based systems such as Pregel [17] and Giraph [3]; systems targeting distributed online processing of *ad hoc* graph queries, such as Horton [23] and Trinity.RDF [28]; and systems such as the Aurelius Graph Cluster which provide a set of technologies targeting the full range of distributed OLTP, querying and analytics [1]. In general, distributed graph processing requires the application of appropriate partitioning and replication strategies to the graph data so as to maximise the *locality* of the processing i.e. minimise the need to ship data between different network nodes.

To determine which architectures, storage and indexing schemes, computational models, algorithms, and partitioning/replication strategies are best suited to which scenarios, new benchmarks are being developed with the aim of providing comparative performance tests for graph data processing [9,20,6].

The recent Big Graph Data Panel at ISWC 2012 [5] discussed several technical challenges arising from big graph data, particularly as relating to the Semantic Web: the need for parallelisation of graph data processing algorithms when the data is too big to handle on one server; the need to understand the performance impact on graph data processing algorithms when the data does not all fit into the total main memory available and to design algorithms explicitly for these scenarios; and the need to find automated methods of handling the heterogeneity, incompleteness and inconsistency between different big graph data sets that need to be semantically integrated in order to be effectively queried or analysed.

In relation to this last point, the explicit modelling and presence in graph data of relationships between entities does provide additional means of identifying and resolving inconsistencies, through following and checking different paths between graph nodes. The explicit representation of relationships in graph data may also have implications on the required levels of consistency in certain usage scenarios, which may be more stringent than for more entity-oriented data given that the temporary presence/absence of an edge in a graph may have a large impact on query results such as reachability or shortest path. Providing the necessary levels of consistency may in turn may have performance implications on the whole workload handling.

Other challenges include: developing heuristics to drive the partitioning of large-scale dynamic graph data for efficient distributed processing, given that the classical graph partitioning problem is NP-Hard [19]; devising new semantics and algorithms for graph pattern matching over distributed graphs, given that the classical subgraph isomorphism problem is NP-complete [16]; developing query formulation and evaluation techniques to assist users querying of complex, dynamic or irregular graph data, such that users may not be aware of its full structure c.f. [11,22]; achieving scalable inferencing over large-scale graph data [24,12]; analysing uncertain graph data [29,27]; enriching of graph data with additional inferred attributes and relationships (e.g. in a social networking setting, inferring information about peoples' interests, knowledge, skills, and social interactions); supporting users' visual exploration of large-scale graph data, and of query and analysis results; and developing algorithms for processing high-volume graph data streams.

2 Panel Discussion

Issues to be discussed in the panel include:

1. What are novel and emerging Use Cases that are generating large volumes of graph data?
2. How does "big" graph data differ from other graph data? Is there a spectrum of increasing volume, velocity and variety; or is there a paradigm shift?
3. Is there a fundamental separation between on-line query processing and large-scale analysis of graph data, or there is there an overlap between them? Is there a need for different architectural approaches for these two aspects of graph data processing or can "one size fit all"?
4. What graph data processing is provided more effectively by special-purpose triple stores or graph databases compared to more general-purpose databases?
5. What processing can be done well with "big" graph data now, what can be done less well, and what are the research challenges and opportunities?

References

1. Aurelius, `http://thinkaurelius.com`
2. Cypher, `http://docs.neo4j.org/chunked/milestone/cypher-query-lang.html`
3. Giraph, `https://github.com/apache/giraph`
4. Gremlin, `https://github.com/tinkerpop/gremlin/wiki/`
5. ISWC 2012 Big Graph Data Panel (2012), `http://semanticweb.com/video-iswcs-big-graph-data-panels_b34112`
6. Linked Data Benchmark Council, `http://www.ldbc.eu/`
7. Neo4j, `http://neo4j.org`
8. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Comput. Surv. 40(1) (2008)
9. Dominguez-Sal, D., Urbón-Bayes, P., Giménez-Vañó, A., Gómez-Villamor, S., Martínez-Bazán, N., Larriba-Pey, J.L.: Survey of graph database performance on the HPC scalable graph analysis benchmark. In: Shen, H.T., Pei, J., Özsu, M.T., Zou, L., Lu, J., Ling, T.-W., Yu, G., Zhuang, Y., Shao, J. (eds.) WAIM 2010. LNCS, vol. 6185, pp. 37–48. Springer, Heidelberg (2010)
10. Faloutsos, C., Kang, U.: Mining billion-scale graphs: Patterns and algorithms. In: SIGMOD 2012, pp. 585–588 (2012)
11. Fernandez, M., Suciu, D.: Optimizing regular path expressions using graph schemas. In: ICDE 1998, pp. 14–23 (1998)
12. Haffmans, W.J., Fletcher, G.H.L.: Efficient RDFS entailment in external memory. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM-WS 2011. LNCS, vol. 7046, pp. 464–473. Springer, Heidelberg (2011)

13. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. In: W3C Recommendation (March 21, 2013)
14. Jin, R., Ruan, N., Dey, S., Xu, J.Y.: Scaling reachability computation on large graphs. In: SIGMOD 2012, pp. 169–180 (2012)
15. Kung, U., Tsourakakis, C.E., Faloutsos, C.: Pegasus: A peta-scale graph mining system - implementation and observations. In: International Conference on Data Mining 2009, pp. 229–238 (2009)
16. Ma, S., Cao, Y., Fan, W., Huai, J., Wo, T.: Capturing topology in graph pattern matching. PVLDB 5(4) (2012)
17. Malewicz, G., Austern, M.H., Bik, A.J.C., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: a system for large-scale graph processing. In: SIGMOD 2010, pp. 135–146 (2010)
18. Martinez-Bazan, N., Aguila-Lorente, M.A., Muntes-Mulero, V., Dominguez-Sal, D., Gomez-Villamor, S., Larriba-Pey, J.L.: Efficient graph management based on bitmap indices. In: IDEAS 2012, pp. 110–119 (2012)
19. Mondal, J., Deshpande, A.: Managing large dynamic graphs efficiently. In: SIGMOD 2012, pp. 145–156 (2012)
20. Morsey, M., Lehmann, J., Auer, S., Ngonga Ngomo, A.-C.: DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 454–469. Springer, Heidelberg (2011)
21. Picalausa, F., Luo, Y., Fletcher, G.H.L., Hidders, J., Vansummeren, S.: A structural approach to indexing triples. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 406–421. Springer, Heidelberg (2012)
22. Poulovassilis, A., Wood, P.T.: Combining approximation and relaxation in semantic web path queries. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 631–646. Springer, Heidelberg (2010)
23. Sarwat, M., Elnikety, S., He, Y., Kliot, G.: Horton: Online query execution engine for large distributed graphs. In: ICDE 2012, pp. 1289–1292 (2012)
24. Urbani, J., Kotoulas, S., Oren, E., van Harmelen, F.: Scalable distributed reasoning using mapReduce. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 634–649. Springer, Heidelberg (2009)
25. Wood, P.T.: Query languages for graph databases. ACM SIGMOD Record 41(1), 50–60 (2012)
26. Yildirim, H., Chaoji, V., Zaki, M.J.: GRAIL: Scalable reachability index for large graphs. PVLDB 3(1-2), 276–284 (2010)
27. Yuan, Y., Wang, G., Wang, H., Chen, L.: Efficient subgraph search over large uncertain graphs. PVLDB 4(11) (2011)
28. Zeng, K., Yang, J., Wang, H., Shao, B., Wang, Z.: A distributed graph engine for web scale RDF data. PVLDB 6(4) (2013)
29. Zou, Z., Gao, H., Li, J.: Mining frequent subgraph patterns from uncertain graph data. TKDE 22(9), 1203–1218 (2010)

Adapting to Node Failure in Sensor Network Query Processing

Alan B. Stokes, Alvaro A.A. Fernandes, and Norman W. Paton

School of Computer Science, University of Manchester,
Manchester M13 9PL, United Kingdom

Abstract. The typical nodes used in mote-level wireless sensor networks (WSNs) are often brittle and severely resource-constrained. In particular, nodes are often battery-powered, thereby making energy depletion a significant risk. When changes to the connectivity graph occur as a result of node failure, the overall computation may collapse unless it is capable of adapting to the new WSN state. Sensor network query processors (SNQPs) construe a WSN as a distributed, continuous query platform where the streams of sensed values constitute the logical extents of interest. Crucially, in the context of this paper, they must make assumptions about the connectivity graph of the WSN at compile time that are likely not to hold for the lifetime of the compiled query evaluation plans (QEPs) the SNQPs generate. This paper address the problem of ensuring that a QEP continues to execute even if some nodes fail. The goal is to extend the lifetime of the QEP, i.e., the period during which it produces results, beyond the point where node failures start to occur. We contribute descriptions of two different approaches that have been implemented in an existing SNQP and present experimental results indicating that each significantly increases the overall lifetime of a query compared with non adaptive approach.

Keywords: Sensor Network Query Processors, Wireless Sensor Networks, Resilience.

1 Introduction

Wireless sensor networks (WSNs) are useful in data collection, event detection and entity tracking applications, among others. In particular, mote-level WSNs are sufficiently inexpensive that one can envisage deploying them to sense at fine granularities both over space and over time. With the low cost, however, come severe resource constraints in terms of energy stock, communication range, computational and storage capabilities, etc. Our focus here is on WSNs comprising static motes of this kind (e.g., [1]).

If one views the WSN as simply an instrument for data collection, one might task the relevant subset of nodes to sense the physical world and send the sensed values, using multi-hop communication paths, towards a base station where all the processing takes place. However, sending all data in raw form to the base station causes more bytes to be transmitted than would be the case if the nodes

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 33–47, 2013.

along the route to the base station were tasked with some of the processing [10]. Since the energy cost of processing data is typically an order of magnitude smaller than the energy cost of transmitting the same data [6], it is more energy-efficient to do as much processing as possible inside the WSN, as this is likely to reduce the number of bytes that are transmitted to the base station.

One approach to in-WSN processing construes the WSN as a distributed database, and the processing task injected into nodes for execution is the evaluation of a query evaluation plan (QEP). In this approach, users specify their data requirements in the form of declarative queries, which the system, called a sensor network query processor (SNQP), compiles into optimized QEPs for injection into the WSN. Through periodic evaluation, a stream of results is returned to the users via the base station.

Many SNQPs have been proposed in the literature, e.g. SNEE [4], TinyDB [9], and AnduIN [7]. These SNQPs often differ in terms, among others, of how much of the required query functionality can be injected into the WSN, how much use they make of distributed query processing techniques (e.g., fragment partitioning, buffering tuples for block transmission, etc.), and how much compile-time knowledge of the WSN state they require in order to produce a QEP. Thus, AnduIN does not inject joins for in-network execution, only QEP leaves, i.e., sensing tasks. AnduIN uses a TCP/IP protocol stack and therefore has no need to know the state of the WSN connectivity graph at compile time. The benefit ensuing from this approach comes at the price of potentially greater energy expenditure (since TCP and IP were not designed with the goal of preserving energy) and of reduced capability to inject functionality into the network (since part of the very scarce program memory has to be assigned to the protocol stack). In contrast, TinyDB is capable of performing limited forms of joins inside the WSN but pushes the entire QEP to every participating node. Again, this is profligate with respect to program memory. TinyDB infers the current connectivity graph from the dissemination of the QEP into the WSN. Finally, SNEE, which we focus on in this paper, pushes very expressive QEPs into the WSN whilst still partitioning the latter into fragments that are as small as possible for each node. However, SNEE neither uses a generic protocol stack nor can it compile the QEP without knowledge of the current connectivity graph.

SNEE does more in-WSN processing than the other SNQPs mentioned above. It generates QEPs that deliver good energy efficiency [4] whilst scheduling for node execution QEP fragment instances that use less memory (partly by not using, and hence not loading, generic protocol stacks) [4] than the other SNQPs mentioned above. To generate QEPs where medium access, routing, and transport are query-specific, the SNEE compiler takes as input (among other metadata) the current connectivity graph. This implies a further, and stronger, assumption, viz., that if the connectivity graph changes (e.g., a node fails) during the lifetime of QEP p, then p may not be optimal for the new topology (and, indeed, p may even be unable to go on running). In other words, maximizing QEP lifetime is dependent on resilience to failure. A SNEE QEP often has its lifetime bounded by the time-to-failure of participating nodes. In practice, not only is node failure assumed to be a common occurrence, the energy stock of participating motes is guaranteed to diminish over time and depletion eventually causes the motes to become non-functional.

SNEE QEPs are therefore particularly brittle: if a participating node fails, poor performance, or even a crash, could ensue. One aspect of poor performance is the lack of adaptation to tuple loss when the corresponding extent draws from a failed node. Such failures lead to partial results for the query. It is, therefore, desirable that, if possible, the QEP is adapted in response to node failure. Another possibility is that the failed node causes the communication graph used by the QEP to partition in such a way that, although all sensed values are flowing out of the leaves, they cannot be used as they fail to reach some downstream operators, i.e., the energy expenditure of the leaves is wasted.

Adaptations aim to minimize information loss and foster compliance with quality of service (QoS) expectations such as maximum delivery rate and constant acquisition rate.

The purpose of adaptations, in the case of this paper, is to maximize the lifetime of the QEP. Since lifetime is influenced by the rate of depletion of energy stocks and since any adaptation will cause some such depletion (i.e., carries an energy overhead cost), adaptations must take into account the time taken to adapt (during which, data will cease to flow) and the energy spent in carrying out the adaptation. Our hypothesis is that the benefit of adapting with a view to significantly increase the QEP lifetime (and, therefore, the amount of data produced) outweighs the cost incurred in adapting.

We compare two strategies that at runtime adapt the QEP in different ways to increase resilience to failure. The first strategy acts as a baseline: it recomputes the entire QEP for the new deployment state, and disseminates the recomputed QEP. This acts like a reset and has high overhead because both the dissemination of a QEP and its writing onto program memory are costly in terms of time and energy expenditure. In contrast, the second strategy identifies adaptations that require the minimal amount of changes to repair the QEP.

The results show that the adaptation costs incurred by the both strategies can lead to significant increases in the lifetime of a QEP.

The rest of the paper is as follows Sec. 2 briefly describes related work. Sec. 3 describes the SNEE SNQP and how the time and energy costs of a QEP are modelled. Sec. 4 describes each strategy. Sec. 5 describes how we experimentally evaluated each one. Sec. 6 draws conclusions.

2 Related Work

Broadly speaking, current SNQPs are not very tolerant of node failure. In TinyDB, the fact that routing trees [9] are constructed during the QEP dissemination process provides some amount of inter-query fault tolerance, as failed nodes do not take part in disseminating the next QEP (which could be a recompilation of the same query) and hence will be disqualified from participating in its evaluation. Also, each node in TinyDB evaluates the entire QEP (i.e., TinyDB makes no attempt to partition the plan into fragments), and, as a result, ignoring a failed node is a sound strategy. Thus, whilst TinyDB is not, strictly speaking, adaptive, it is, to a certain degree, resilient to some forms of node failure. However, tuple transmission is not globally scheduled (as it is in SNEE), so there is no way of estimating how many tuples might be lost as a result of failed nodes.

The SmartCIS project [8] builds upon TinyDB with a particular goal (among others) of supporting fault-tolerant routing trees via multi-path transmissions. This approach incurs energy overheads in verifying that current paths are correct and in searching for new correct ones.

AnduIN has no specific mechanism for fault tolerance. In contrast with both TinyDB and SNEE, which compile into TinyOS [5], AnduIN compiles into Contiki [3]. The difference is relevant in our context because, unlike TinyOS, Contiki provides a TCP/IP-based communication protocol stack. Thus, AnduIN benefits from the robust routing and transport properties built into TCP/IP. The drawback is that TCP/IP incur much greater overheads (and take up more memory footprint) than the minimalistic, query-specific protocols used by TinyDB and SNEE. Some of these overheads stem from the need to maintain up-to-date connectivity paths as well as from the need to send acknowledgement packets. As to memory occupancy, TCP/IP implementations will take up space and will also claim more memory for such structures as routing tables. By reducing the memory on the nodes that can be allocated to the QEP, there is a reduction in how much processing can be shipped to the WSN and how much memory can be used buffering and blocked transmission, both features that are energy-saving. AnduIN does not adapt to failure of acquisition nodes.

Our prior work [11] explored logical overlays which use redundant nodes (fortuitous or planned) within the network to achieve resilience to node failure using clusters of equivalent nodes.

3 Technical Context

SNEE aims to generate energy-efficient QEPs. The compilation/optimization process takes as input a SNEEql query (as exemplified in Fig. 1), QoS expectations (not shown in the figure) in the form of a desired acquisition rate (i.e., the frequency at which sensing takes place) and a maximum delivery time (i.e., an upper bound on the acceptable amount of time between data being acquired and being reflected in the emitted results), and the following kinds of metadata: (1) the current connectivity graph, which describes the (cost-assigned) communication edges in the WSN; (2) the logical schema for the query, which describes the available logical extents over the sensing modalities in the WSN; (3) the physical schema for the query, which describes which physical nodes contribute data to which logical extent, and which node acts as base station; (4) statistics about nodes (e.g., available memory and energy stocks); and (5) cost-model parameters (e.g., unit costs for sleeping, sensing, processing, and communicating) [2]. The example query takes two streams, one stemming from sensors in a field, the other from sensors in a forest. It joins them on the condition that light levels are higher in the field than in the forest and emits onto the output stream the matching values and the ids of the nodes that generated them. The intuition behind the query is that if light levels in the forest are higher than in the open field, then one might suspect that a forest fire has started.

Fig. 2 shows the SNEE (compilation/optimization) stack. As a distributed query optimizer, it uses a two-phase approach. The single-site phase (Steps 1-3 in Fig. 2) comprises the classical steps needed to compile and optimize a query for centralized execution. The outcome is the physical-algebraic form (PAF) for the query, where each operator has been given its execution order and assigned a concrete algorithm. The multi-site phase (Steps 4-7 in Fig. 2) turns the PAF into a distributed algebraic form (DAF) for the query by making decisions that are specific to in-WSN execution. These include deciding on a routing tree R, on fragment instance allocation along the routing tree captured as a DAF D and on timing the activities in the nodes (switching from QEP fragment evaluation to communication and so on) in the form of an agenda A. A final step converts the triple $\langle R, D, A\rangle$ into a set of per-node nesC/TinyOS source files, which are then compiled into binary form. This is what we refer to as the executable QEP.

In more detail, Step 4 in Fig. 2 generates a routing tree (RT) for the query as an approximation of a Steiner tree, e.g., the one in Fig. 3(a) for our example query. Each vertex is a sensor node; an edge denotes that the two nodes can communicate; the arrow denotes the direction of communication; double-line circles denote the sink or else nodes that do sensing; single-line nodes do processing or communication or both. Recall that a Steiner tree is a minimum spanning tree (and hence likely to be energy-efficient) that necessarily includes a given set of nodes. In our case, these are the leaves (i.e., the acquisition nodes) and the root (i.e., the base station).

Step 5 in Fig. 2 decides which fragment instances to place for execution in which node. This partitions the PAF into fragment instances and assigns the latter to RT nodes with a view to conserving energy by reducing the number of tuples that need to be transmitted. The resulting DAF for the example query is shown in Fig. 3(b). Dashed boxes define fragment boundaries; the list in curly brackets at the bottom-right corner (below the fragment identifier) denotes how many instances of that fragment there are and in which nodes they run. The fragment containing the deliver operator runs on the sink node, the fragment instances containing the acquisition operators run on the leaf nodes and the remaining fragment instances are assigned to run on Node 1 because it is, amongst the nodes through which all tuple streams required for the join flow, the hop-count closest to the leaves. We call such nodes, *confluence nodes.*

```
Logical Schema:  field  (id, time, temp, light); forest (id, time, temp, light);

Physical Schema: field: {N6, N9}; forest: {N7}; sink: {N8}

Q: SELECT RSTREAM c.id, c.light, f.id, f.light   FROM field[NOW] c, forest[NOW] f
   WHERE  c.light < f.light
```

Fig. 1. Example Query, Logical/Physical Schemas

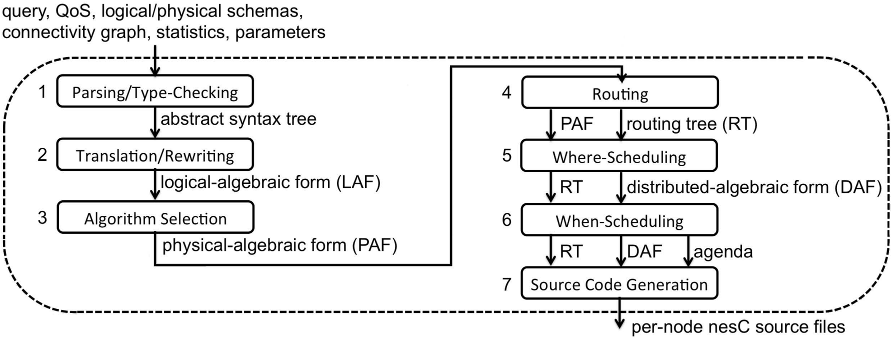
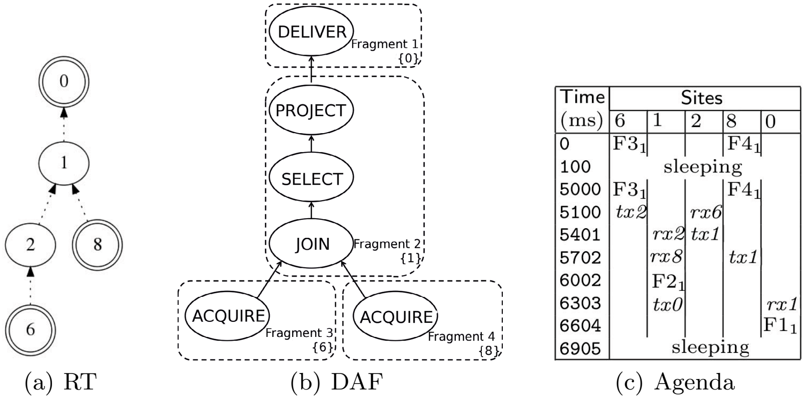

query, QoS, logical/physical schemas,
connectivity graph, statistics, parameters
1 Parsing/Type-Checking
abstract syntax tree
2 Translation/Rewriting
logical-algebraic form (LAF)
3 Algorithm Selection
physical-algebraic form (PAF)
4 Routing
PAF
routing tree (RT)
5 Where-Scheduling
RT
distributed-algebraic form (DAF)
6 When-Scheduling
RT
DAF
agenda
7 Source Code Generation
per-node nesC source files

Fig. 2. The SNEE Compilation Stack

Step 6 in Fig. 2 decides when to execute the different tasks in each participating node. These decisions are represented as an agenda, i.e., a matrix where rows denote points in time in the query evaluation cycle, columns denote participating nodes, and the content of each cell defines the task scheduled for that node at that time. The agenda for the example query is shown in Fig. 3(c). Fragments are identified by number as in Fig. 3(b), with subscripts denoting fragment instances; the notation *txn* (resp., *rxn*) denotes that that node at that time is transmitting to (resp., receiving from) node *n*; a row labelled *sleeping* denotes the fact that, for that slot, all the nodes in the WSN are idle. In the process of deciding on an agenda, SNEE also determines how much buffering of tuples can be done on the nodes with the memory remaining from fragment instance allocation. By being governed by an agenda, a SNEE QEP implements a simple form of TDMA (time-division multiple-access to channels). Whilst this is often economical provided that the estimation models are accurate (and [2] shows that the ones used in SNEE are), any changes to the timing of the operators or transmissions requires the agenda to be recomputed and hence the QEP to be recompiled and propagated into the WSN.

Step 7 in Fig. 2 takes the RT from Step 4, the DAF from Step 5, and the agenda from Step 6 to produce a set of per-participating-node source files. Compiling these files yields the binaries that, together, comprise the QEP. For an in-depth description of the SNEE compilation stack and data structures, see [4].

We note that, as described in [2], SNEE makes intensive use of the empirically-validated analytical cost models for energy, memory and time expenditure computed over the RT, DAF and agenda for a query. For example, in SNEE, we can estimate the energy and time cost of running a QEP fragment instance on any given node per agenda execution cycle. Such capabilities allow us to estimate the time that a QEP will run before a given node fails due to energy depletion.



Time (ms)	Sites 6	1	2	8	0
0	$F3_1$			$F4_1$	
100	sleeping				
5000	$F3_1$			$F4_1$	
5100	*tx2*		*rx6*		
5401		*rx2*	*tx1*		
5702		*rx8*		*tx1*	
6002		$F2_1$			
6303		*tx0*			*rx1*
6604					$F1_1$
6905	sleeping				

(a) RT (b) DAF (c) Agenda

Fig. 3. Example Inputs to Code Generation

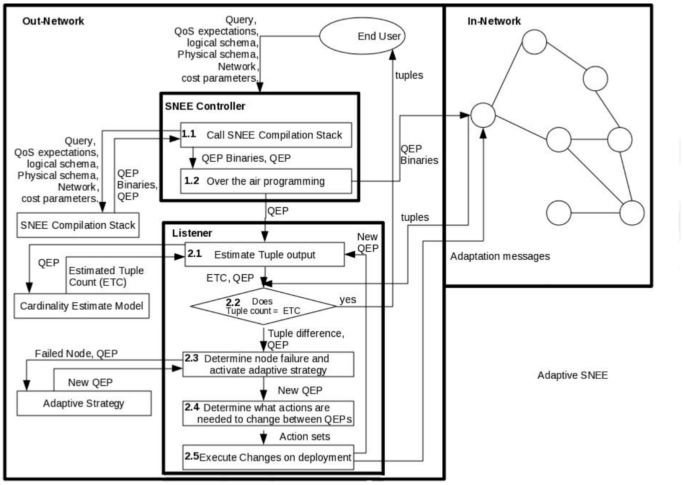


Fig. 4. The adaptive infrastructure

4 Adaptation Strategies

Figure 4 provides an overview of the Stages 1.1 and 1.2 represent the compile time actions executed by the SNEE SNQP; Once stage 1.2 has been executed, there would be an operating QEP that would be producing output tuples.

The problem of handling node failure can then be broken down into the problems of monitoring this QEP for node failures, determining how to respond to the failure and finally responding to the failure. In Figure 4 Stages 2.1 to 2.5 shows how the SNEE infrastructure has been modified to include an adaptive control loop that monitors the network by using models to assess non intrusively if the network has experienced a node failure (Stages 2.1 and 2.2 in Figure 4). When a node failure has been detected, an adaptive strategy is executed to determine how to repair the QEP to recover from the failure (represented as stage 2.3 in Figure 4). The strategy returns a new QEP which is then broken down into a set of adaptation actions that together can change the currently running QEP into the new QEP during Stage 2.4 of Figure 4. These actions are:

1. Reprogramming the node completely (giving it a fresh binary to execute).
2. Redirecting the destination to which tuples are being sent.
3. Deactivating a node so that it no longer participates in the QEP.
4. Activating a node so that it can start participating in the QEP.
5. Shifting the node's agenda by a specific time period, thereby making the node change when it executes its operators and engages in data transport.

The reprogramming of nodes is the most expensive of the above actions in terms of both energy and time, as node binaries imply a large number of data packets to transmit, and these packets need to be stored in flash memory. The other four actions only require one message to be sent to the nodes in question as these changes require no reprogramming of the node. The time and energy used whilst the WSN is adapting is time and energy that the QEP is not devoting to processing tuples for the end user, and so should be as small as possible.

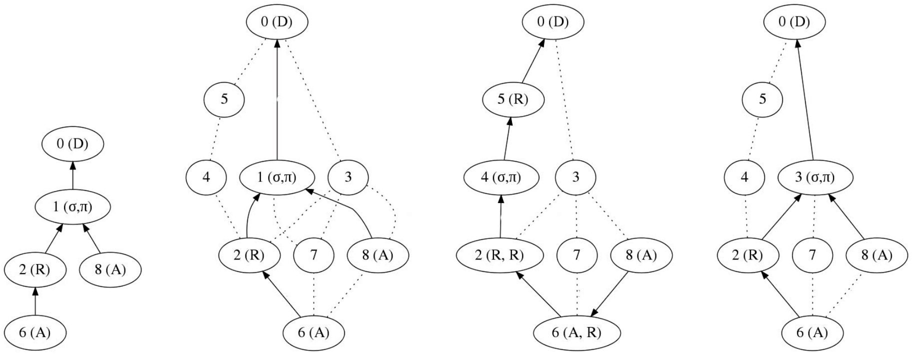
These action messages are then sent into the network in Stage 2.5 of Figure 4. This adaptation cycle repeats if necessary until either the query has finished executing (e.g., it was required to run for a fixed amount of time), or the network experiences a failure from which it cannot recover.

The problem of repairing a QEP from node failure is split into two problems. The first is to reroute tuples around the failed nodes. The second problem is to reassign operators that were previously running on the failed node to new nodes whilst still meeting Quality of Service (QoS) expectations.

We propose two different strategies. The *global strategy* tries to adapt whilst minimising the new QEP's execution cost. The *regional repair strategy* tries to adapt with the minimal amount of runtime reprogramming.

We assume that each strategy has the same node failure detection procedure which in our case is based on significant discrepancies between the estimated and the observed number of tuples returned by the QEP given a defined selectivity value for each operator. These estimates are obtained by an analytical model. Finally, we also assume that the metadata representing the WSN has already been updated to reflect the failure.

In the next two subsections we define how each strategy adapts to node failure. For the sake of clarity, we use a different representation of the DAF by first overlaying it over the RT as shown in Figure 5(a). This allows us to show the extra communication operators and the physical layout of the DAF which is not easily apparent in a DAF. As these strategies try to exploit nodes not currently being used by the QEP, we overlay this representation on top of the entire deployment (as shown in Figures 5(b),5(c) and 5(d)). This gives us a simple graphical example of the different DAFs generated by the strategies.



(a) Original DAF overlayed on the RT

(b) Original DAF overlay on the Deployment

(c) Global's DAF overlay on the Deployment

(d) Regional's DAF overlay on the Deployment

Fig. 5. Adaptations after the failure of Node 1

Figure 5(b) shows the original DAF with tuples being acquired at node 6 and 8 which are then forwarded to node 1 where select and project operators are executed. The results from these operators are forwarded to node 0 which delivers the tuples out of the network. Nodes 3,4,5 and 7 are not participating in the current QEP. Each communication path available is annotated with the energy cost of transmitting tuples through it.

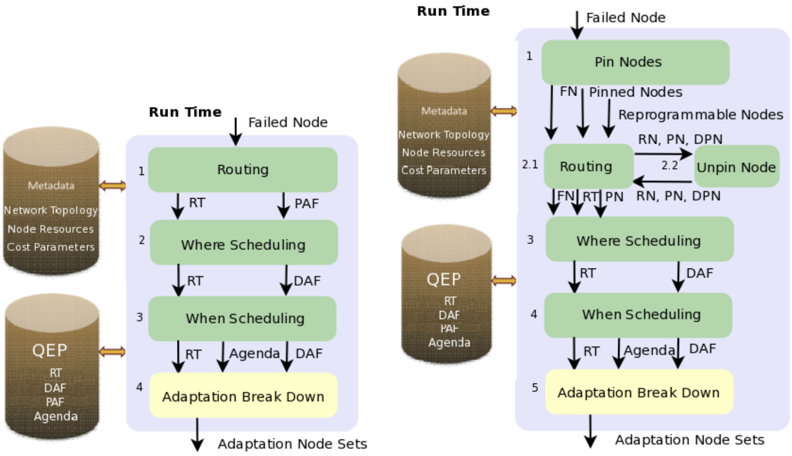
4.1 Global Strategy

The global strategy seeks to generate the least energy expensive QEP for the new state of the network. This strategy uses all available information about the network and re-optimises over the entire network.

The global strategy is the simplest of the two strategies as the first three stages in the strategy's execution stack (represented in Figure 6(a)) are identical to the multi-site phase of the SNEE compilation stack in Figure 2. At the end of stage three, a new RT, DAF and Agenda will have been computed, in this case resulting in the DAF depicted in Figure 5(c).

The downside to the global strategy is that its likely to result in reprogramming of several nodes, as any change in the fragments in a node requires a complete reprogramming of the node.

Adaptation Break Down. Stage 4 of the global execution stack compares the new and old QEP data structures and looks for changes between them. In Figure 5(c) nodes 2,4,5 and 6 are reprogrammed, due to having no operators on them (as in nodes 4 and 5) or having the incorrect fragments (as in nodes 6 and 2). Node 8 only requires a redirection action to change its parent node's id from the failed node to node 6.



(a) Global (b) Regional Repair

Fig. 6. Each Strategies Execution Stack

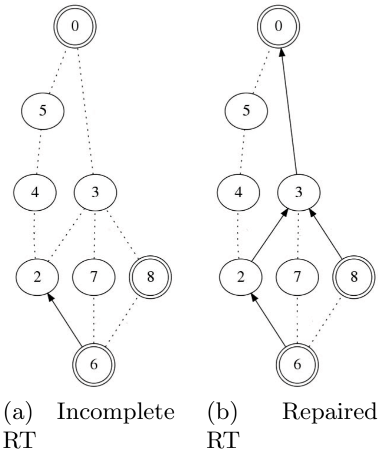
These actions are then used in the adaptation stage, where pre-defined messages are forwarded from the base station to the nodes in question. For reprogramming, the messages contain the entire new node binary, which is sent down the RT to the node in question, where it is installed into flash memory. For redirection, activation, deactivation and temporal adjustment, one message is sent communicating changes in variables in the binary, an example of which is a change of the node id that acts as its parent.

4.2 Regional Repair Strategy

The regional repair strategy tries to reduce the amount of run time reprogramming by exploring a collection of possible changes in the neighbourhood of the failed node aiming to retain unchanged as much as possible of the original plan. Therefore it essentially seeks to repair the current plan rather than to identify a brand new one. It aims to derive a new QEP that requires the least amount of reprogramming to transition from the old to the new QEP, thereby reducing the time and energy overheads incurred whilst the WSN is adapting. The downside to this strategy is that the repaired QEP may be more expensive than the one that would have been produced by the global strategy.

The strategy works in five stages as depicted in the regional repair execution stack in Figure 6(b). In each of the following five subsections we explain how each step operates for the example depicted in Figure 5(b).

Pin Nodes. The pinning stage (numbered 1 in Figure 6(b)) works to reduce the scope of the routing algorithm by removing nodes from consideration from the connectivity graph for routing decisions. The goal is to avoid disrupting the rest of the QEP by starting with nodes which do not currently participate within the



(a) Incomplete RT (b) Repaired RT

Fig. 7. Comparison between the original, incomplete and repaired RT's

QEP. This is done by traversing the connectivity graph and locating any node which is in the original RT and is not a failed node. These nodes are said to be pinned. The rest are reprogrammable. The original RT is updated to reflect the failed node, resulting in a disconnection in the original RT, as in Figure 7(a). The pinned nodes, failed nodes and the reprogrammable set are then used as inputs for the routing stage. In this example nodes 2,6,8 and 0 are pinned, and node 1 is removed from the RT.

Routing and Unpinning. Stage 2.1 in Figure 6(b) attempts to repair the RT with the least amount of reprogramming. This is done by first locating sections of the RT where the failed node and any unpinned nodes (initially an empty set) communicate with each other. Each of these sections is repaired independently. In the example only one section is located, viz., node 1.

Each section then gives rise to a *Steiner tree* problem where the inputs of the leaf nodes of the section (node 2 and 8) and the parent of the root node of the section (node 0) become mandatory nodes (represented as double circled nodes in Figure 7). A heuristic algorithm given in [6] is used over the union of the reprogrammable and unpinned sets to compute a Steiner tree, in this case forcing a route through node 3, as node 3 can communicate directly with nodes 0, 2 and 8. Each section's Steiner tree solution is then integrated into the disconnected RT to create a new RT as represented in Figure 7(b).

If the heuristic algorithm fails to find a tree to connect the mandatory nodes in a section, then Stage 2.2 is initiated in order to choose one of the pinned nodes to become unpinned and therefore reprogrammable. This is done by selecting the parent of the root node in the problem section. If no parent node is available (because the algorithm has travelled to the sink node) then a node is chosen at random. At this point, Stage 2.1 is restarted, this cycle continues until all sections have a corresponding Steiner tree solution.

Where Scheduling. Stage 3 takes the new RT and generates a DAF by placing all non-attribute sensitive pinned QEP fragments onto the pinned nodes. Any remaining QEP fragments (in this case, the one containing the project and select operators originally placed on node 1) are placed on the most applicable site (site 3) determined by the where scheduling phase discussed in Section 3. This results in the DAF shown in Figure 5(d).

When Scheduling and Adaptation Breakdown. Stage 4 behaves exactly as the SNEE when scheduler in Figure 2 and the same comparison stage as mentioned in the global strategy is executed, resulting in node 3 being reprogrammed and nodes 2 and 8 tuples being redirected to node 3 instead of node 1.

5 Experimental Evaluation

This section presents experimental evidence to test the hypothesis that the benefit of adapting with a view to increasing the QEP lifetime (and, therefore, the amount of data produced) outweighs the cost incurred in adapting. All experiments were carried out using the Avrora [12] simulator.

We began by generating a set of 30 synthetic WSN topologies, with corresponding physical and logical schemas with the same generator used in [4]. Each topology comprises 30 nodes with random connectivity to a subset of other nodes.[1] For a third of the topologies, `SELECT *` queries were generated over the available sensed extent. For another third of the topologies, aggregation queries were generated. For the final third, join queries were generated. In all cases, we have set the QoS expectations as follows: the acquisition rate is a tuple every 10 seconds and the maximum delivery time is 600 seconds.

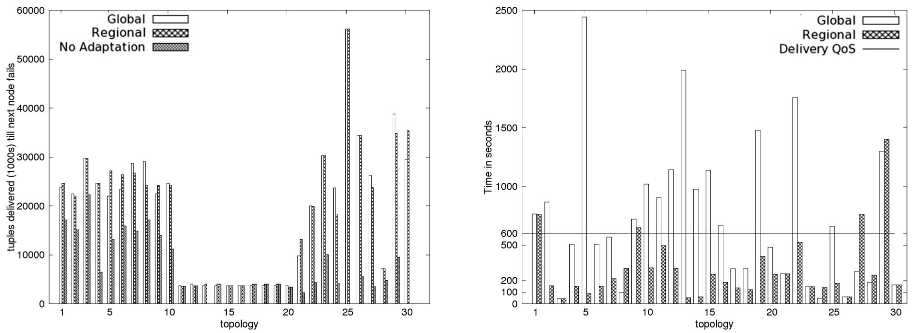
For each topology, query, logical and physical schemas, we used SNEE to generate the corresponding, initial QEP.

When simulating the failure of a node, we take the following into account: (a) we prefer confluence nodes both because they are, by definition, crucial for the correctness of the query, and because they tend to have a higher workload than other types of node and are, therefore, more likely to fail in practice; and (b) we exclude acquisition nodes because it allows us to evaluate the net benefit of adaptive responses between functionally-equivalent plans.

Results and Observations. To evaluate how the energy drain from the adaptation affects the lifetime (and, therefore, tuple delivery) of the network, we estimate how many agenda cycles can be completed successfully by each node in the RT and use the lowest value as the maximum number of successful agenda execution cycles for the QEP. From the estimated lifetime we can infer the maximum number of tuples that can be delivered to the base station by multiplying the lifetime by the calculated maximum tuple generation per agenda execution cycle.

These results are shown in Figure 8(a). The following observations can be made from these results:

[1] We ran experiments over larger topologies, but no new conclusions emerged so we have omitted the results to save space.



(a) Number of Tuples Delivered Covering the Different Types of Queries

(b) Length of Time taken to Adapt For Each Strategy

Fig. 8. Results

1. As all the strategies use a derivative of the routing algorithm used by the SNEE compiler, there is a risk that confluence nodes which have not failed will be used to transmit reprogramming packets down to nodes. This puts more pressure on these nodes resulting in them failing earlier than expected and therefore delivering fewer tuples. Examples of this are topologies 5, 7, 25 and 30.
2. Most of the time, adapting to node failure results in an extended QEP lifetime. This is due to the re-distribution of operators to different nodes in the network, that often have more energy than the nodes that participated in the original OTA programming, therefore resulting in a extended lifetime and delivered more tuples.
3. When comparing the numbers of tuples delivered to the original QEP without an adaptation, each strategy results in a substantial benefit in total tuple delivery. Because aggregation operators reduce all input tuples into a single output one, in aggregation queries there is no noticeable difference in tuple delivery unless the entire data stream is disconnected. We point out that accuracy of the resulting tuple may now be severely reduced (e.g. a count or an average).
4. The difference in tuple gain between the strategies is only approximately 1 to 4 % with neither outperforming the other on a regular basis. This shows that for one adaptation, the energy used up by the adaptations is discounted by the regular cost of the executing QEP. We note that given more node failures during the QEPs execution may result in more significant differences between the strategies, and is left for future work.

The time the network is adapting to the node failure is time that it is not producing results, and therefore is important to consider as it will result in gaps within the output steam. This motivated us to observe the time period taken to execute the changes within the network, show in Figure 8(b). The following observations can be made from these results:

1. In most cases the global strategy produces adaptations which incur a significantly higher time cost than the regional strategy. This means that using the global strategy is likely to generate more down time and therefore larger gaps within the output.
2. In most cases the regional strategy executes all of its changes within one agenda cycle, and therefore it could be possible to repair and reproduce the lost result before the next agenda cycle is executed by exploiting the time period at the bottom of an agenda, where all nodes are doing nothing, to retransmit the lost packets.

In summary, we have provided experimental evidence that adapting to node failure as it occurs gives rise to significant benefits in terms of additional lifetime that averages of approx 4.00% for select * queries, 1.47% for aggregation queries, and 27.64% for join queries. We also determined that regional changes could result in less downtime for making the adaptations in relation to doing a global reset.

6 Conclusion

This paper has described two strategies for recovering from node failure in SNQP and makes the following contributions:

1. We have described approaches that adapt to node failure where one strategy re-optimises the QEP to the new state of the network, and the other strategy repairs the in-network QEP with as little in-network reprogramming as possible.
2. We have reported on empirical performance evaluation of the two strategies. The experiments consistently show that adapting to node failure is beneficial in relation to the number of tuples that are delivered to the base station by extending the lifetime of the QEP.

Acknowledgements. This work has been funded by the Engineering and Physical Sciences Research Council and we are grateful for this support.

References

1. Telosb mote platform, `http://www.willow.co.uk/TelosB_Datasheet.pdf`
2. Brenninkmeijer, C.Y.A., Galpin, I., Fernandes, A.A.A., Paton, N.W.: Validated cost models for sensor network queries. In: DMSN (2009)
3. Dunkels, A., Grönvall, B., Voigt, T.: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: LCN, pp. 455–462 (2004)
4. Galpin, I., Brenninkmeijer, C.Y.A., Gray, A.J.G., Jabeen, F., Fernandes, A.A.A., Paton, N.W.: SNEE: a query processor for wireless sensor networks. Distributed and Parallel Databases 29(1-2), 31–85 (2011)
5. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D.E., Pister, K.S.J.: System Architecture Directions for Networked Sensors. In: ASPLOS, pp. 93–104 (2000)

6. Holger, K., Willig, A.: Protocols and Architectures for Wireless Sensor Networks. John Wiley and Sons (June 2005)
7. Klan, D., Karnstedt, M., Hose, K., Ribe-Baumann, L., Sattler, K.-U.: Stream engines meet wireless sensor networks: cost-based planning and processing of complex queries in AnduIn. Distributed and Parallel Databases 29(1-2) (2011)
8. Liu, M., Mihaylov, S.R., Bao, Z., Jacob, M., Ives, Z.G., Loo, B.T., Guha, S.: Smart CIS: integrating digital and physical environments. SIGMOD Rec., pp. 48–53
9. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst. 30(1), 122–173 (2005)
10. Pottie, G.J., Kaiser, W.J.: Wireless integrated network sensors. Commun. ACM 43(5), 51–58 (2000)
11. Stokes, A.B., Fernandes, A.A.A., Paton, N.W.: Resilient sensor network query processing using logical overlays. In: MobiDE, pp. 45–52 (2012)
12. Titzer, B., Lee, D.K., Palsberg, J.: Avrora: scalable sensor network simulation with precise timing. In: IPSN, pp. 477–482 (2005)

Loop Elimination for Database Updates

Vadim Savenkov[1], Reinhard Pichler[1], and Christoph Koch[2]

[1] Vienna University of Technology
{savenkov,pichler}@dbai.tuwien.ac.at
[2] École Polytechnique Fédérale de Lausanne
christoph.koch@epfl.ch

Abstract. The additional expressive power of procedural extensions of query and update languages come at the expense of trading the efficient set-at-a-time processing of database engines for the much less efficient tuple-at-a-time processing of a procedural language. In this work, we consider the problem of rewriting for-loops with update statements into sequences of updates which do not use loops or cursors and which simultaneously carry out the action of several loop iterations in a set-at-a-time manner. We identify idempotence as the crucial condition for allowing such a rewriting. We formulate concrete rewrite rules for single updates in a loop and extend them to sequences of updates in a loop.

1 Introduction

To enhance the expressive power of SQL for querying and modifying data, the SQL standard proposes SQL/PSM as a Turing complete procedural extension of SQL. Most relational database management systems provide their own procedural programming language such as PL/pgSQL of PostgreSQL, PL/SQL of Oracle, SQL PL of IBM's DB2, Transact-SQL in Microsoft SQL Server, etc. The key feature of these extensions of SQL is to allow the definition of loops for iterating over relations with a cursor and to "parameterize" so to speak the action in the loop body by the current tuples of these relations. The additional expressive power however comes at the expense of trading the efficient set-at-a-time processing of SQL for the much less efficient tuple-at-a-time processing of a procedural language. For the sake of optimizing updates, the question naturally arises if a given sequence of updates necessarily has to be realized by a loop containing these updates or whether it would be possible to achieve the same effect with a sequence of simple updates that do not use loops or cursors.

In this paper we restrict ourselves to *for*-loops with updates. Our goal is to provide rewrite rules that allow one to transform for-loops with update statements into a sequence of simple updates that simultaneously carry out the action of several loop iterations in a set-at-a-time manner. To this end, we will first introduce an update language which we find convenient for our investigations and point out how update statements of this form can be represented in (standard) SQL. We then identify a crucial property of updates as a sufficient condition for the elimination of for-loops containing update statements, namely the *idempotence* of updates, i.e., applying the same update twice or more often yields the

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 48–61, 2013.

same result as a single application of the update. Based on this condition, we shall define rewrite rules for unnesting a single update and also several successive updates in a for-loop. The elimination of nested loops with updates is thus also covered by successively applying the rewrite rules to each loop - starting with the innermost one.

Update optimization is an old topic in database research. A problem statement similar to ours was considered by Lieuwen and DeWitt in [5], who provided rules for optimizing for-loop statements in the database programming language O++. There, the authors focus on flattening the nested loops. In contrast, our approach allows for complete elimination of loops and replacing them with relational-style update commands. This problem has been also considered in [1] in the context of uncertain databases. The results in the present paper extend that work: in particular, we consider update commands in which arbitrary attributes can be referenced on the right-hand side of equalities in the *set*-clause, whereas in [1] only constants are supported.

Our transformation relies on the *idempotence* of update operations, which can be easily tested: the operation is idempotent if repeating it twice or more times has the same effect as applying it only once. The importance of the idempotence property for update optimization for the task of incremental maintenance of materialized views [4], has been identified in [3]. More recently, idempotent operations have been found useful also in a broader setting in the area of distributed systems [2,6]. Efficient and block-free methods of failure recovery are essential in distributed environments. The idempotence property ensures that such repeated evaluation is safe and does not change the semantics of a program.

Organization of the Paper and Summary of Results. In Section 2, we introduce a simple update language, which is convenient for our investigations, and establish its connection to (standard) SQL. In Section 3 we present our rewrite rule for eliminating a for-loop with a single update statement inside. Clearly, this rewrite rule is also applicable to nested loops starting with the innermost loop. The crucial condition for the applicability of our rewrite rule is the idempotence of the update. In Section 4, we formulate a sufficient condition for the elimination of for-loops with more than one update inside and present an appropriate extension of our rewrite rule.

2 Update Language

Suppose that we want to update some relation R whose schema $\mathbf{sch}(R)$ is given as $\mathbf{sch}(R) = \{A_1, \ldots, A_m\}$. In this paper, we restrict ourselves to updates which can be defined by a relation U with $\{A_1, \ldots, A_m, A'_1, \ldots, A'_m\} \subseteq \mathbf{sch}(U)$ in the following sense: the tuples affected by such an update are $T = \{r \mid \exists u \in U$, s.t. $r.A_1 = u.A_1 \wedge \ldots \wedge r.A_m = u.A_m\}$, i.e., $T = R \ltimes U$. The new values to which the attributes $\bar{A} = (A_1, \ldots, A_m)$ of each tuple $r \in T$ are modified are defined by the components $\bar{A}' = (A'_1, \ldots, A'_m)$ of the corresponding tuple in U, i.e.: each $r \in T$ is modified to $\pi_{\bar{A}'}(\sigma_{U.\bar{A}=r}(U))$. Clearly, there must exist a functional dependency

$U.\bar{A} \rightarrow U.\bar{A}'$ to make sure that every tuple in T is mapped to precisely one value $U.\bar{A}'$. This leads to the definition of the following language of update statements:

Definition 1. *Let R and U be relations with $\textbf{sch}(R) = \{A_1, \ldots, A_m\}$ and $\{A_1, \ldots, A_m, A'_1, \ldots, A'_m\} \subseteq \textbf{sch}(U)$. Then the "update defined by U" is denoted as*

$$\boxed{\text{update R set } \bar{A} = U.\bar{A}' \text{ from } U \text{ where } R.\bar{A} = U.\bar{A};}$$

Such an update is called well-defined *if there exists a functional dependency $U.\bar{A} \rightarrow U.\bar{A}'$. In this case, the effect of this update is to replace each tuple r in $R \ltimes U$ by the uniquely determined value $U.\bar{A}'$, s.t. $r.\bar{A} = U.\bar{A}$.*

Note that the above definition imposes no restriction on the nature of the relation U. In particular, U may itself be defined by some query. In this case, the value of U immediately before updating R is considered as fixed throughout the update process. This is in line with the transactional semantics of SQL updates, i.e., changes made by an update are not visible to the update itself before the end of the update operation.

The proposed syntax is general enough to cover many practical cases. In particular, the updates of the form "update R set $\bar{A} = \bar{c}$ where ϕ", considered in [1], can be captured easily: Let Q_ϕ denote the semi-join query returning the tuples of R that have to be updated. In order to write the above update in the form: update R set $\bar{A} = U.\bar{A}'$ from U where $R.\bar{A} = U.\bar{A}$; we have to define the relation U. Suppose that $\textbf{sch}(R) = \{A_1, \ldots, A_m, B_1, \ldots B_n\}$, which we abbreviate as $\bar{A}, \bar{B}$. Likewise, we write $\bar{A}', \bar{B}'$ to denote $\{A'_1, \ldots, A'_m, B'_1, \ldots B'_n\}$. Then $U(\bar{A}, \bar{B}, \bar{A}', \bar{B}')$ is defined by the following query (in logic programming notation):

$U(\bar{A}, \bar{B}, \bar{c}, \bar{B})$:- $R(\bar{A}, \bar{B}), Q_\phi$;

We give some further simple examples of updates below:

Example 1. Consider a relation R with attributes (A_1, A_2). An update operation that swaps the two attributes of R can be defined as

$$\text{update R set } \bar{A} = U.\bar{A}' \text{ from } U \text{ where } R.\bar{A} = U.\bar{A};$$

such that $U(A_1, A_2, A'_1, A'_2)$ is defined by the following query:

$U(A_1, A_2, A_2, A_1)$:- $R(A_1, A_2)$.

Now suppose that A_1, A_2 have numerical values. Moreover, suppose that we want to apply the above update only to tuples r in R where A_1 is an even number and $A_1 < A_2$ holds. Then we just have to modify the definition of relation U, namely:

$U(A_1, A_2, A_2, A_1)$:- $R(A_1, A_2)$, $A_1 < A_2$, $A_1 \bmod 2 == 0$.

More generally, suppose that R contains m attributes $(A_1, \ldots, A_m)$ and we want to swap the first two of them. Then $U(A_1, \ldots, A_m, A'_1, \ldots, A'_m)$ is defined as follows:

$U(A_1, A_2, A_3, \ldots, A_m, A_2, A_1, A_3, \ldots, A_m)$:- $R(A_1, \ldots, A_m)$. □

We conclude this section by describing a translation of updates formulated in our syntax to standard SQL. Consider an update of the form

$$\text{update R set } \bar{A} = U.\bar{A}' \text{ from } U \text{ where } R.\bar{A} = U.\bar{A};$$

where $\bar{A}$ denotes the attributes $\{A_1, \ldots, A_m\}$ of R. This update can be rewritten as follows:

```
update $R$ set
$A_1$ = (select $A'_1$ from $U$ where $R.\bar{A} = U.\bar{A}$)
...
$A_m$ = (select $A'_m$ from $U$ where $R.\bar{A} = U.\bar{A}$)
where exists (select * from $U$ where $R.\bar{A} = U.\bar{A}$).
```

where we write $R.\bar{A} = U.\bar{A}$ to abbreviate the condition $R.A_1 = U.A_1$ and ... and $R.A_m = U.A_m$. If the DBMS supports the extended update syntax (like 'update from' in PostgreSQL), then the SQL update statement becomes more concise:

```
update $R$ set $A_1 = A'_1$, ..., $A_m = A'_m$
from $U$
where $R.\bar{A} = U.\bar{A}$.
```

Of course, in simple cases, the relation U does not have to be defined explicitly (e.g., as a view), as the following example illustrates:

Example 2. Consider relations R, S, P with $\mathbf{sch}(R) = \{A_1, A_2, A_3\}$, $\mathbf{sch}(S) = \{B_1, B_2\}$, and $\mathbf{sch}(P) = \{C_1, C_2\}$. Let an update be defined by the relation $U(A_1, A_2, A_3, A'_1, A'_2, A'_3)$, where U is defined as follows:

$$U(A_1, A_2, A_3, A_1, A'_2, A'_3) \text{ :- } S(A_2, A'_2), P(A_3, A'_3), A'_2 < A'_3.$$

Intuitively, S defines the update of the second component of R and P defines the update of the third component of R. Moreover, these updates may only be applied if the new value for the second component of R is less than for the third one. In SQL we get:

```
update $R$
set
   $A_2$ = (select $S.B_2$ as $A'_2$ from $S$, $P$
            where $S.B_1 = R.A_2$ and $P.C_1 = R.A_3$ and $S.B_2 < P.C_2$),
   $A_3$ = (select $P.C_2$ as $A'_3$ from $S$, $P$
            where $S.B_1 = R.A_2$ and $P.C_1 = R.A_3$ and $S.B_2 < P.C_2$)
where exists (select * from $S$, $P$
            where $S.B_1 = R.A_2$ and $P.C_1 = R.A_3$ and $S.B_2 < P.C_2$).
```

If the DBMS supports the extended update syntax, then the above update statement can be greatly simplified to:

```
update $R$
set $A_2 = S.B_2$, $A_3 = P.C_2$
from $S$, $P$
where $S.B_1 = A_2$ and $P.C_1 = A_3$ and $S.B_2 < P.C_2$
```

□

```
for ($\$t$ in $Q$){update R set $\bar{A} = U[\$t].\bar{A}'$ from $U[\$t]$ where $R.\bar{A} = U[\$t].\bar{A}$};
$\vdash$   update R set $\bar{A} = V.\bar{A}'$ from $V$ where $R.\bar{A} = V.\bar{A}$;
    s.t. $V = \bigcup_{t \in Q} U[t]$.
```

Fig. 1. Unnesting update programs

3 Loop Elimination

Recall from Definition 1 that, in this paper, we are considering updates defined by some relation U. Now suppose that an update occurs inside a loop which iterates over the tuples in some relation Q. Hence, in general, the update relation U depends on the current tuple t of Q. We thus write $U[t]$ to denote the value of U for a given tuple t of Q. In a loop over Q, the relation U is parameterized so to speak by the tuples in Q. We thus write $U[\$t]$ to denote the family of relations U that we get by instantiating the variable $\$t$ by the tuples t in Q. We thus have to deal with loops of the following form:

```
for ($\$t$ in $Q$){update R set $\bar{A} = U[\$t].\bar{A}'$ from $U[\$t]$ where $R.\bar{A} = U[\$t].\bar{A}$};
```

where $\mathbf{sch}(R) = \{A_1, \ldots, A_m\}$. Moreover, for every instantiation of $\$t$ to a tuple t over the schema $\mathbf{sch}(Q)$, $U[\$t]$ yields a relation whose schema contains the attributes $\{A_1, \ldots, A_m, A'_1, \ldots, A'_m\}$. The relation resulting from instantiating $\$t$ to t is denoted as $U[t]$. The semantics of the above loop is the following: *for each value $\$t$ in Q, peform the update of R using the update relation $U[\$t]$ according to Definition 1.*

Of course, updates may also occur inside nested loops. We thus get statements of the following form:

```
for ($\$t_1$ in $Q_1$) {
  for ($\$t_2$ in $Q_2$) {
    ...
      for ($\$t_n$ in $Q_n$) {update R
                              set $\bar{A} = U[\$t_1, \ldots, \$t_n].\bar{A}'$
                              from $U[\$t_1, \ldots, \$t_n]$
                              where $R.\bar{A} = U[\$t_1, \ldots, \$t_n].\bar{A}$}...}};
```

such that, for every instantiation of $\$t_1, \ldots, \$t_n$ to tuples $t_1, \ldots, t_n$ over the schemas $\mathbf{sch}(Q_1), \ldots, \mathbf{sch}(Q_n)$, $U[\$t_1, \ldots, \$t_n]$ yields a relation whose schema contains the attributes $\{A_1, \ldots, A_m, A'_1, \ldots, A'_m\}$. The relation resulting from instantiating $\$t_1$ to t_1, ..., $\$t_n$ to t_n is denoted as $U[t_1, \ldots, t_n]$.

For unnesting updates, it suffices to provide a rule for transforming a single for-loop with update into an update statement without loop. In case of nested loops, this transformation has to be applied iteratively starting with the innermost for-loop. Such a rule can be found in Fig. 1. It is put into effect in the following example:

Example 3. Consider relations Department and Employee: $\mathbf{sch}(\mathsf{Department}) = \{\mathsf{dept\_id}, \mathsf{bonus}\}$, $\mathbf{sch}(\mathsf{Employee}) = \{\mathsf{empl\_id}, \mathsf{dept\_id}, \mathsf{base\_salary}, \mathsf{compensation}\}$. Using logic programming notation, we define the relations Q and U:

$$Q(\mathit{Dept\_id}, \mathit{Bonus}) \text{ :- } \mathsf{Department}(\mathit{Dept\_id}, \mathit{Bonus})$$

$$\begin{aligned}&U(\mathit{Empl\_id}, \mathit{Dept\_id}, \mathit{Base\_sal}, \mathit{Comp},\\ &\quad \mathit{Empl\_id}, \mathit{Dept\_id}, \mathit{Base\_sal}, \mathit{Comp}')[\$t] \text{ :- } \mathit{Comp}' = \mathit{Base\_sal} \cdot (1 + \$t.\mathsf{bonus})\end{aligned}$$

The following update loop increases the compensation of all employees:

```
for($t in Q )
    update Employee set compensation = U[$t].compensation′ from U[$t]
    where U[$t].empl_id = Employee.empl_id and
              U[$t].dept_id = Employee.dept_id and
              U[$t].base_salary = Employee.base_salary and
              U[$t].compensation = Employee.compensation
```

For the sake of readability, in the *set*-clause of the update command we omit the assignments to the attributes which are not modified in $U[\$t]$. Applying the unnesting rule from Fig. 1, the update loop can be rewritten as the following command:

```
    update Employee set compensation = V.compensation′ from V
    where V.empl_id = Employee.empl_id and
              V.dept_id = Employee.dept_id and
              V.base_salary = Employee.base_salary and
              V.compensation = Employee.compensation
```

Here, V is obtained from $U[\$t]$ by taking a join of $U[\$t]$ with Q and replacing $\$t$ in the body of U with Q.

$$\begin{aligned}V(\mathit{Empl\_id}, \mathit{Dept\_id}, \mathit{Base\_sal}, \mathit{Comp}, \mathit{Empl\_id}, \mathit{Dept\_id}, \mathit{Base\_sal}, \mathit{Comp}') \text{ :- }&\\ \mathit{Comp}' = \mathit{Base\_sal} \cdot (1 + Q.\mathit{Bonus}),&\\ Q(\mathit{Dept\_id}, \mathit{Bonus})&\end{aligned}$$

It is easy to see that the above expression for V expresses exactly the one in Fig. 1, namely $V = \bigcup_{t \in Q} U[t]$. This expression can be further simplifed as

$$\begin{aligned}V(\mathit{Empl\_id}, \mathit{Dept\_id}, \mathit{Base\_sal}, \mathit{Comp}, \mathit{Empl\_id}, \mathit{Dept\_id}, \mathit{Base\_sal}, \mathit{Comp}') \text{ :- }&\\ \mathit{Comp}' = \mathit{Base\_sal} \cdot (1 + \mathsf{Department}.\mathit{Bonus}),&\\ \mathsf{Department}(\mathit{Dept\_id}, \mathit{Bonus})&\end{aligned}$$

□

Remark 1. Note that to ensure that the update is well-defined, it might be necessary to inspect the particular instance of the join between the updated table and the update relation U. However, if certain integrity constraints are present in the schema, such inspection can be spared, since the desired key dependency may be inferred from the definition of the update relation U and existing schema constraints. For instance, for the update relation $U[\$t](\bar{A}, \bar{A}')$ in Example 3, the dependency $\bar{A} \to \bar{A}'$ has to be checked. Since the only modified attribute of Employee is compensation, it suffices to check the functional dependency $\bar{A} \to \mathsf{compensation}'$, where $\bar{A}$ denotes the first four attributes of $U[\$t]$. Note that $\mathsf{compensation}'$ is determined by base_salary and Department.bonus. Moreover, note that empl_id and dept_id are respective primary keys in the Employee and Department tables. Then, also the functional dependency empl_id → bonus holds in the relation Employee ⋈ Department. Therefore, the functional dependency $\mathsf{empl\_id} \to \mathsf{compensation}'$ holds in $U[\$t]$ and thus the respective update is well-defined irrespective of the database instance.

The following theorem gives a sufficient correctness criterion for the loop elimination from Fig. 1.

Theorem 1. *Let Q and R be relations with $\boldsymbol{sch}(Q) = \{B_1, \dots, B_\ell\}$ and $\boldsymbol{sch}(R) = \{A_1, \dots, A_m\}$. Moreover, let $U[\$t]$ be a parameterized relation with $\boldsymbol{sch}(U[\$t]) = \{A_1, \dots, A_m, A_1', \dots, A_m'\}$ and $\boldsymbol{sch}(\$t) = \boldsymbol{sch}(Q)$. Finally, suppose that, for every $\$t \in Q$, the update of R defined by $U[\$t]$ is well-defined (cf. Definition 1).*

The rewrite rule in Fig. 1 is correct (i.e., the update resulting from the rewrite rule application has the same effect on the database as the loop), if the following conditions are fulfilled.

1. *In V, the functional dependency $V.\bar{A} \to V.\bar{A}'$ holds, i.e, the update of R by V is well-defined.*
2. *The relation $\rho_{\bar{A} \leftarrow \bar{A}'}(\pi_{\bar{A}'}(V)) \bowtie \pi_{\bar{A},\bar{A}'}(V)$ contains only tuples which fulfill the selection criterion $\sigma_{\bar{A}=\bar{A}'}$.*
3. *The relation R is not used in the definition of $U[\$t]$, i.e., $U[\$t]$ is not modified by the update.*

Remark 2. The second condition in the above theorem reads as follows: Consider all tuples in $\pi_{\bar{A}'}(V)$. They constitute a superset of the values that may possibly occur as the result value of some update step. The renaming $\rho_{\bar{A} \leftarrow \bar{A}'}$ and the join with $\pi_{\bar{A},\bar{A}'}(V)$ computes the result value (for arbitrary tuple $t \in Q$) if the update is applied to the same row of R again. The second condition thus requires that applying the update again must not alter the value anymore. In other words, the second condition imposes a strong kind of idempotence, i.e., if r' is the result obtained from updating $r \in R$ in the loop iteration according to some $t \in Q$, then the update of r' for any tuple $t' \in Q$ must not alter r'. Many real-world updates are idempotent: for instance, the commands setting attributes equal to constants, or looking up new values using join expressions, provided that the attributes used for the look-up are not affected by the update (cf. Example 3).

The third condition above means that we are considering loops with updates defined by relations U whose value is not modified inside the loop. Note that this restriction is quite realistic since otherwise the semantics of the loop might easily depend on the concrete order in which the tuples t of the "outer relation" Q are processed.

Note that if the update relation $U[\$t](\bar{A}, \bar{A}')$ is such that all attributes in $\bar{A}$ are either equal to the corresponding attributes in $\bar{A}'$ or not bound in the body of U, the second condition is fulfilled trivially (cf. Example 3). If also the first condition of updates to be well-defined is enforced by the schema constraints (as described in Remark 1), then the applicability of the transformation in Fig. 1 can be checked statically, that is, without inspecting the actual instance.

Proof (Theorem 1). We first introduce some useful notation: Suppose that a tuple $r \in R$ is affected by the update in the i-th iteration of the loop, i.e., $r \in R$ is replaced by some tuple r'. Of course, it may happen that this result tuple r' itself is affected by the update in the j-th iteration of the loop with $j > i$. For this proof, it is convenient to think of the tuples $r \in R$ as equipped with an additional attribute id, which serves as a unique identifier of the tuples and which is never altered by any update. Hence, by identifying every tuple $r \in R$ with its unique id, we may distinguish between a tuple $id(r)$ and its value r. In particular, the updates only change the values of the tuples in R, while R contains always the same set of tuples.

Now let $T = \{t_1, \ldots, t_n\}$ denote the tuples in Q, s.t. the loop processes the tuples in Q in this (arbitrarily chosen) order. For every $i \in \{1, \ldots, n\}$, let Q_i and V_i be defined as $Q_i = \{t_1, \ldots, t_i\}$ and $V_i = \bigcup_{t \in Q_i} U[t]$. Clearly, it suffices to prove the following claim in order to prove the theorem:

Claim A. For every $i \in \{1, \ldots, n\}$, the update defined by the loop

$$\text{for } (\$t \text{ in } Q_i)\{\text{update R set } \bar{A} = U[\$t].\bar{A}' \text{ from } U[\$t] \text{ where } R.\bar{A} = U[\$t].\bar{A}\};$$

has the same effect on the database as the update

$$\text{update R set } \bar{A} = V_i.\bar{A}' \text{ from } V_i \text{ where } R.\bar{A} = V_i.\bar{A};$$

We proceed by induction on i with $i \in \{1, \ldots, n\}$:

"$i = 1$" In this case, we have $Q_1 = \{t_1\}$. Thus, the above for-loop is iterated exactly once and the corresponding update of R is defined by $U[t_1]$. On the other hand, we have $V_1 = U[t_1]$. Hence, the update defined by V_1 is precisely the same as the update in the (single iteration of) the loop.

"$(i-1) \to i$" By definition, $Q_i = Q_{i-1} \cup \{t_i\}$ and $V_i = V_{i-1} \cup U[t_i]$. We first show that the tuples of R affected by the first i iterations of the above loop coincide with tuples of R affected by the unnested update defined by V_i. In the second step, we will then show that the affected tuples of R are mapped to the same value by the loop-updates and by the unnested update.

Let $r \in R$ with identifier id. We observe the following equivalences: r is affected by an update in the first i iterations of the loop $\Leftrightarrow$ there exists a $j \leq i$ and a tuple $u \in U[t_j]$, s.t. $r.\bar{A} = u.\bar{A}$ holds $\Leftrightarrow$ r is affected by the update defined by $V_i = \bigcup_{t \in Q_i} U[t]$.

As for the value of the tuple $r \in R$ with identifier id after the i iterations of the for-loop respectively after the update defined by V_i, we distinguish 3 cases:

Case 1. Suppose that r is affected by the first $i - 1$ iterations of the loop but the resulting tuple r' (which still has the same identifier id) is not affected by the i-th iteration. By the induction hypothesis, the updates carried out by the first $i - 1$ loop iterations and the update defined by V_{i-1} have the same effect on r, namely they both modify r to r'. By assumption, this value is unchanged in the i-th loop iteration. On the other hand, since $V_{i-1} \subseteq V_i$ and, by condition 1 of the theorem, the updated defined by V (and, therefore also by $V_i \subseteq V$) is well-defined. Hence, the update defined by V_i has the same effect on r as V_{i-1}.

Case 2. Suppose that r is affected by the i-th iteration of the loop for the first time. The update by the i-th loop iteration is defined by $U[t_i]$. On the other hand, $U[t_i] \subseteq V_i$ and, by condition 1 of the theorem, the updated defined by V (and, therefore also by $V_i \subseteq V$) is well-defined. Hence, both in the loop and in the unnested update, the tuple r is modified to the value r' according to the update defined by $U[t_i]$.

Case 3. Suppose that r is affected by the first $i-1$ iterations of the loop and the resulting tuple r' (which still has the same identifier id) is again affected by the i-th iteration. By the induction hypothesis, the updates carried out by the first $i-1$ loop iterations and the update defined by V_{i-1} have the same effect on r. Let the resulting value in both case be denoted as r'. Since $V_{i-1} \subseteq V_i$ and V_i is well-defined, the update defined by V_i also modifies r to r'. It remains to show that the i-th iteration of the loop does not alter r' anymore. Suppose that r' is modified to r'' by the update defined by $U[t_i]$. Clearly, $r' \in \pi_{\bar{A}'}(V)$. Moreover, $(r', r'') \in U[t_i] \subseteq V$ and, therefore, $(r', r'') \in \rho_{\bar{A} \leftarrow \bar{A}'}(\pi_{\bar{A}'}(V)) \bowtie \pi_{\bar{A},\bar{A}'}(V)$. Hence, $r' = r''$ by condition 3 of the theorem. □

If we want to apply the unnesting according to Theorem 1 to updates inside nested loops, we have to start from the innermost loop. Suppose that the nested loop looks as follows:

```
for ($\$t_1$ in $Q_1$) {
  for ($\$t_2$ in $Q_2$) {
    ...
      for ($\$t_n$ in $Q_n$) {update R
                              set $\bar{A} = U[\$t_1, \ldots, \$t_n].\bar{A}'$
                              from $U[\$t_1, \ldots, \$t_n]$
                              where $R.\bar{A} = U[\$t_1, \ldots, \$t_n].\bar{A}$}...}};
```

In this case, $U[\$t_1, \ldots, \$t_n]$ plays the role of $U[\$t]$ in Theorem 1 and the conditions of Theorem 1 have to be fulfilled for all possible instantiations of the parameters $\$t_1, \ldots, \$t_{n-1}$ over the corresponding domains.

The following example illustrates the problems which could arise if no idempotence condition were required:

Example 4. Consider the program

for ($\$t$ in Q){update R set $A = U[\$t].A'$ from $U[\$t]$ where $R.A = U[\$t].A$};

such that R, Q, and $U[\$t]$ are relations with $\mathbf{sch}(R) = \{A\}$, $\mathbf{sch}(Q) = \{B_1, B_2\}$, and $\mathbf{sch}(U[\$t]) = \{A, A'\}$. Suppose that the update relation $U[\$t]$, which is parameterized by the tuple $\$t$, is defined as follows:

$U[\$t](A, A')$:- $A = \$t.B_1$, $A' = \$t.B_2$.

In other words, we consider a loop which is controlled by the tuples of Q, s.t. each tuple $t \in Q$ defines an update on R, namely if $t.B_1$ coincides with some entry $(A) \in R$, then (A) is replaced by $t.B_2$.

Table 1. Non-idempotence of updates

$\mathbf{R.A_1}$	$\mathbf{Q.B_1}$	$\mathbf{Q.B_2}$	$\mathbf{Q'.B_1}$	$\mathbf{Q'.B_2}$
1	1	2	1	2
2	2	3	2	2

First, suppose that the relations R and Q are given in Table 1. In this case, the result of the update loop depends on the order in which the elements of Q are processed: if the tuple $(1, 2) \in Q$ is chosen first, then both tuples in R are updated to (3) (the first tuple of R is processed by each iteration: 1 is replaced with 2 and then further replaced with 3 at the second iteration). On the other hand, if the tuple $(2, 3) \in Q$ is processed first by the loop, then R is updated to $\{(2), (3)\}$ by the entire loop.

Clearly, a loop whose result depends on the order in which the tuples of Q are selected is usually not desired. In this case, condition 2 of Theorem 1 is violated and, hence, the rewriting of Fig. 1 is not allowed. Indeed, condition 2 requires that the relation $(\rho_{B_1 \leftarrow B_2}(\pi_{B_2}(Q)) \ltimes_{B_1=B_2} Q)$ consist of tuples with two equal columns, which is not the case.

Now suppose that we use the relation Q' instead of Q. Then the condition 2 of Theorem 1 is satisfied. Indeed, with any order of selecting the tuples of Q' in the loop, R gets modified to $\{(2), (2)\}$. □

As could be seen in the previous example, the violation of condition 2 of Theorem 1 may indicate an undesired behavior of the loop. However, a non-idempotent behavior of an update inside a loop is not always undesired. For instance, consider the following variation of the update in Example 3 increasing the salary of each employee who has been participating in a successfully finished project.

Example 5. Consider the schema of Example 3 extended with the relations Project and EmployeeProject with $\mathbf{sch}(\mathsf{EmployeeProject}) = \{\mathsf{empl\_id}, \mathsf{proj\_id}\}$ and $\mathbf{sch}(\mathsf{Project}) = \{\mathsf{proj\_id}, \mathsf{status}\}$. Suppose that the employee's compensation grows depending on the number of successful projects she has been working in. For instance, the following statement can be used to update the *Employee* table to reflect such a policy:

$$\begin{aligned} Q(Empl\_id, Proj\_id) \text{:-}\ & \mathsf{EmployeeProject}(Empl\_id, Proj\_id),\\ & \mathsf{Project}(Proj\_id, \text{'success'}) \end{aligned}$$

$$\begin{aligned} U(Empl\_id, Dept\_id, Base\_sal, Comp, Empl\_id, Dept\_id, Base\_sal, Comp')[\$t] \text{ :-}& \\ Empl\_id = \$t.\mathsf{empl\_id}&,\\ \mathsf{Department}(Dept\_id, Bonus)&,\\ Comp' = Comp \cdot (1 + Bonus)& \end{aligned}$$

for ($\$t$ in Q)$\{\mathrm{up}_1; \dots \mathrm{up}_n\}$;
$\vdash$ for ($\$t$ in Q)$\{\mathrm{up}_{j_1}; \dots \mathrm{up}_{j_\alpha}\}$; for ($\$t$ in Q)$\{\mathrm{up}_{j_{\alpha+1}}; \dots \mathrm{up}_{j_n}\}$;
s.t. up_i denotes the update of R_i by $U_i[\$t]$, i.e., up_i is of the form
update R_i set $\bar{A}_i = U_i[\$t].\bar{A}'_i$ from $U_i[\$t]$ where $R_i.\bar{A}_i = U_i[\$t].\bar{A}_i$;

Fig. 2. Unnesting update programs

```
for($\$t$ in $Q$ )
   update Employee set compensation = $U[\$t]$.compensation' from $U[\$t]$
   where $U[\$t]$.empl_id = Employee.empl_id and
            $U[\$t]$.dept_id = Employee.dept_id and
            $U[\$t]$.base_salary = Employee.base_salary and
            $U[\$t]$.compensation = Employee.compensation
```

The update is well-defined but not idempotent: the incremented compensation depends on the previous compensation. The restriction of Theorem 1 is not desirable here. □

Different special cases like this leave a space for refining the preconditions for loop elimination. E.g., the update predicate can be iterated some reasonable number of times to check if the update becomes deterministic at some point reachable by the update loop.

4 Loops with More Than One Update

In this section, we consider the case of two (or more updates) inside a loop. If these updates operate on different relations R_1 and R_2, then the loop can obviously be decomposed into two loops with a single update inside. More generally, we define the following rewrite rule.

Theorem 2. *Let Q, $R_1, \dots, R_n$ be relations with $\boldsymbol{sch}(Q) = \{B_1, \dots, B_\ell\}$ and $\boldsymbol{sch}(R_i) = \bar{A}_i = \{A_{i1}, \dots, A_{im_i}\}$ for $i \in \{1, \dots, n\}$, and let let $U_1[\$t], \dots, U_n[\$t]$ be parameterized relations with $\boldsymbol{sch}(U_i[\$t]) = \{A_{i1}, \dots, A_{im_i}, A'_{i1}, \dots, A'_{im_i}\}$ and $\boldsymbol{sch}(\$t) = \boldsymbol{sch}(Q)$. Moreover, suppose that, for every $\$t \in Q$, the update of R_i defined by $U_i[\$t]$ is well-defined (cf. Definition 1).*

The rewrite rule in Fig. 2 is correct (i.e., the two loops resulting from the rewrite rule application have the same effect on the database as the original loop), if the following conditions are fulfilled.

1. *The set $\{1, \dots, n\}$ is partitioned into two sets $J_1 = \{j_1, \dots, j_\alpha\}$ and $J_2 = \{j_{\alpha+1}, \dots, j_n\}$, s.t. the two sequences of indices $(j_1, \dots, j_\alpha)$ and $(j_{\alpha+1}, \dots, j_n)$ are arranged in increasing order.*
2. *$\{R_s \mid s \in J_1\}$ and $\{R_s \mid s \in J_2\}$ are disjoint.*

Proof. Let $T = \{t_1, \ldots, t_N\}$ denote the tuples in Q, s.t. the loop processes the tuples in Q in this (arbitrarily chosen) order. For every $k \in \{1, \ldots, N\}$, let $T_k = \{t_1, \ldots, t_k\}$. We claim that, for every $k \in \{1, \ldots, N\}$ the following rewriting is correct:

$$\boxed{\begin{array}{l} \text{for (\$}t\text{ in }T_k)\{\mathrm{up}_1;\ \ldots\mathrm{up}_n\};\\ \vdash\quad \text{for (\$}t\text{ in }T_k)\{\mathrm{up}_{j_1};\ \ldots\mathrm{up}_{j_\alpha}\};\ \ \text{for (\$}t\text{ in }T_k)\{\mathrm{up}_{j_{\alpha+1}};\ \ldots\mathrm{up}_{j_n}\}; \end{array}}$$

The correctness of this decomposition of the loop into two loops can be proved by an easy induction argument which uses the facts that the relations $U_1, \ldots, U_n$ are never modified inside these loops and the updates in the two resulting loops operate on different relations R_s with $s \in J_1$ and $R_{s'}$ with $s' \in J_2$. Hence, there is no interdependence between the updates in the two resulting loops. □

From now on, we may concentrate on the case that all updates in a loop operate on the same relation R. Below we define a rewrite rule for contracting two updates of the same relation R to a single update. By repeating this rewrite step, any number of updates of the same relation R can be rewritten to a single update of R.

Theorem 3. *Let R, U_1, and U_2 be relations with $\boldsymbol{sch}(R) = \{A_1, \ldots, A_m\}$ and $\boldsymbol{sch}(U_i) = \{A_1, \ldots, A_m, A'_1, \ldots, A'_m\}$ for $i \in \{1, 2\}$ and suppose that the update defined by each U_i is well-defined. Moreover, let U'_i be defined as follows:*

$$\begin{aligned}
U'_i(X_1, \ldots, X_m, X'_1, \ldots, X'_m) &\coloneq U_i(X_1, \ldots, X_m, X'_1, \ldots, X'_m).\\
U'_i(X_1, \ldots, X_m, X_1, \ldots, X_m) &\coloneq R(X_1, \ldots, X_m),\\
&\ \mathit{not}\ U_i(X_1, \ldots, X_m, \_, \ldots, \_).\\
U'_i(X_1, \ldots, X_m, X_1, \ldots, X_m) &\coloneq U_1(\_, \ldots, \_, X_1, \ldots, X_m),\\
&\ \mathit{not}\ U_1(X_1, \ldots, X_m, \_, \ldots, \_).\\
U'_i(X_1, \ldots, X_m, X_1, \ldots, X_m) &\coloneq U_2(\_, \ldots, \_, X_1, \ldots, X_m),\\
&\ \mathit{not}\ U_2(X_1, \ldots, X_m, \_, \ldots, \_).
\end{aligned}$$

Finally, we define V with $\boldsymbol{sch}(V) = \{A_1, \ldots, A_m, A'_1, \ldots, A'_m\}$ as follows:

$$\begin{aligned}
V(X_1, \ldots, X_m, X'_1, \ldots, X'_m) &\coloneq U'_1(X_1, \ldots, X_m, Y_1, \ldots, Y_m),\\
&\ U'_2(Y_1, \ldots, Y_m, X'_1, \ldots, X'_m).
\end{aligned}$$

Then the rewrite rule in Fig. 3 is correct, i.e., the update of R defined by V is also well-defined and has the same effect on the database as the two successive updates of R by U_1 and U_2.

Remark 3. Note that there are two possibilities why the update of a relation R defined by some relation U leaves a value combination $(a_1, \ldots, a_m)$ of the attributes $(A_1, \ldots, A_m)$ unchanged: either U does not contain a row, s.t. the first m columns

{update R set $\bar{A} = U_1.\bar{A}'$ from U_1 where $R.\bar{A} = U_1.\bar{A}$;
update R set $\bar{A} = U_2.\bar{A}'$ from U_2 where $R.\bar{A} = U_2.\bar{A}$}
$\vdash$ update R set $\bar{A} = V.\bar{A}'$ from V where $R.\bar{A} = V.\bar{A}$;
s.t. V is defined as in Theorem 3.

Fig. 3. Contracting two updates

coincide with $(a_1, \ldots, a_m)$; or U contains the row $(a_1, \ldots, a_m, a_1, \ldots, a_m)$. Intuitively, the latter case makes the identity mapping for the tuple $(a_1, \ldots, a_m)$ in R explicit. The intuition of each relation U_i' in the above theorem is that it defines exactly the same update of R as U_i. The only difference between U_i' and U_i is that U_i' makes all possible identity mappings explicit.

Proof. Let r be an arbitrary tuple in r and suppose that r is modified to r' by the update defined by U_1 (of course, $r' = r$ if r is not affected by this update). Moreover, let r' be further modified to r'' by the update defined by U_2. Then either $r \notin \pi_{\bar{A}}(U_1)$ or $(r, r') \in U_1$. In either case, $(r, r') \in U_1'$. Likewise, we may conclude that $(r', r'') \in U_2'$ holds. Hence, also $(r, r'') \in U_1' \bowtie U_2' = V$ holds. Note that the value of r' is uniquely determined by r. This is due to the definition of U_1' and to the fact that U_1 is well-defined. Likewise, the value of r'' is uniquely determined by r'. Hence, the update defined by V is well-defined. Moreover, it indeed modifies r to r''. □

In total, we define the following algorithm for unnesting updates in for-loops:

1. In case of nested for-loops, start with the innermost loop.
2. If a loop contains several updates affecting more than one relation, then replace the for-loop by several successive for-loops each updating a single relation (by iteratively applying the rule of Fig. 2).
3. If a loop contains several updates which all affect the same relation, then replace this sequence of updates by a single update (by iteratively applying the rule of Fig. 3).
4. Replace a loop with a single update by an update without loop (by applying the rule of Fig. 1).

From a program optimization point of view, also partial unnesting via our techniques may lead to much more efficient queries – even if complete unnesting is not always possible (due to the conditions which are required for our transformation rules to be correct).

5 Conclusion

We have considered the problem of unnesting relational updates with cursors and replacing them with simpler, purely relational update expressions. The full set of our rewrite rules can handle loops with one or multiple update statements.

Unnesting and loop elimination can drastically simplify the database program, making it truly declarative and thus more readable and accessible for optimization through appropriate components of the database engine.

Our technique crucially relies on the idempotence of the update operation. Reasonable in most cases, in some situations this requirement can be too restrictive, as discussed in Section 3 (see Example 5). More fine-grained optimization techniques of update loops, relaxing the idempotence requirement where appropriate, as well as more elaborate techniques of splitting loops with multiple updates are left for future work.

Acknowledgements. The research of V. Savenkov and R. Pichler is supported by the Austrian Science Fund (FWF): P25207-N23. The work of C. Koch is supported by Grant 279804 of the European Research Council.

References

1. Antova, L., Koch, C.: On APIs for probabilistic databases. In: QDB/MUD, pp. 41–56 (2008)
2. de Kruijf, M.A., Sankaralingam, K., Jha, S.: Static analysis and compiler design for idempotent processing. SIGPLAN Not. 47(6), 475–486 (2012)
3. Gluche, D., Grust, T., Mainberger, C., Scholl, M.H.: Incremental updates for materialized OQL views. In: Bry, F. (ed.) DOOD 1997. LNCS, vol. 1341, pp. 52–66. Springer, Heidelberg (1997)
4. Gupta, A., Mumick, I.S. (eds.): Materialized views: techniques, implementations, and applications. MIT Press, Cambridge (1999)
5. Lieuwen, D.F., DeWitt, D.J.: A transformation-based approach to optimizing loops in database programming languages. SIGMOD Rec. 21(2), 91–100 (1992)
6. Ramalingam, G., Vaswani, K.: Fault tolerance via idempotence. In: Proc. of POPL 2013, pp. 249–262. ACM, New York (2013)

Append Storage in Multi-Version Databases on Flash

Robert Gottstein[1], Ilia Petrov[2], and Alejandro Buchmann[1]

[1] TU-Darmstadt, Databases and Distributed Systems, Germany
{gottstein,buchmann}@dvs.tu-darmstadt.de
[2] Reutlingen University, Data Management Lab, Germany
ilia.petrov@reutlingen-university.de

Abstract. Append/Log-based Storage and Multi-Version Database Management Systems (MV-DBMS) are gaining significant importance on new storage hardware technologies such as Flash and Non-Volatile Memories. Any modification of a data item in a MV-DBMS results in the creation of a new version. Traditional implementations, physically stamp old versions as invalidated, causing in-place updates resulting in random writes and ultimately in mixed loads, all of which are suboptimal for new storage technologies. Log-/Append-based Storage Managers (LbSM) insert new or modified data at the logical end of log-organised storage, converting in-place updates into small sequential appends. We claim that the combination of multi-versioning and append storage effectively addresses the characteristics of modern storage technologies.

We explore to what extent multi-versioning approaches such as Snapshot Isolation (SI) can benefit from Append-Based storage, and how a Flash-optimised approach called SIAS (Snapshot Isolation Append Storage) can improve performance. While traditional LbSM use coarse-grain page append granularity, SIAS performs appends in tuple-version granularity and manages versions as simply linked lists, thus avoiding in-place invalidations.

Our experimental results instrumenting a SSD with TPC-C generated OLTP load patterns show that: a) traditional LbSM approaches are up to 73% faster than their in-place update counterparts; b) SIAS tuple-version granularity append is up to 2.99x faster (IOPS and runtime) than in-place update storage managers; c) SIAS reduces the write overhead up to 52 times; d) in SIAS using exclusive append regions per relation is up to 5% faster than using one append region for all relations; e) SIAS I/O performance scales with growing parallelism, whereas traditional approaches reach early saturation.

1 Introduction

Multi-Version Database Management Systems (MV-DBMS) and Log/Append-based Storage Managers (LbSM) are gaining significant importance on new storage hardware technologies such as Flash and Non-Volatile Memories. Compared to traditional storage such as HDD or main memory new storage technologies have fundamentally different characteristics. I/O patterns have major influence on their performance and endurance: especially overwrites and (small) random writes are significantly more expensive than a sequential write.

MV-DBMS create new versions of data items once they are modified. Treating old and new versions differently provides a mechanism to leverage some of the properties of

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 62–76, 2013.

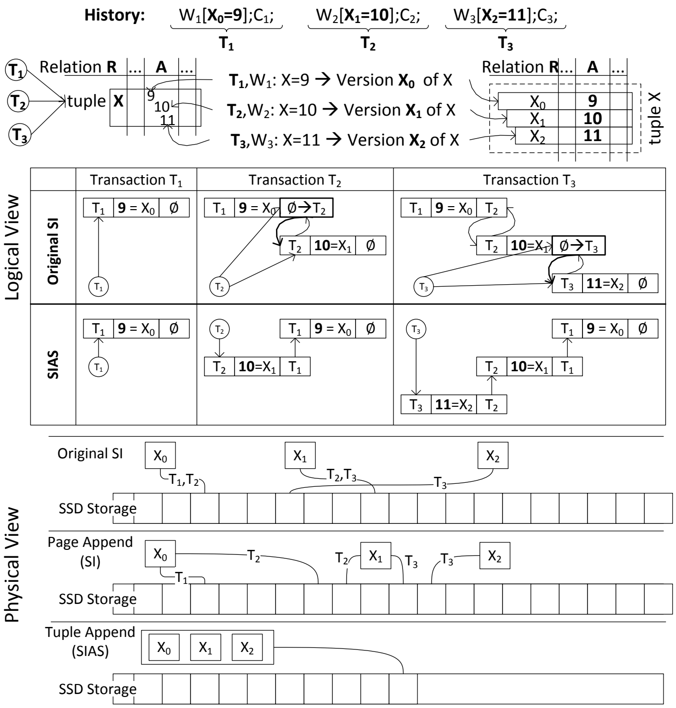


Fig. 1. Version handling

new storage, such as fast reads and low latency. However, HDD (read) optimised implementations such as Snapshot Isolation (SI) invalidate old versions physically in-place as successor versions are created, resulting in random writes and mixed load which is suboptimal for new storage technologies. Additionally they do not leverage read/write asymmetry. Log/Append-based storage managers (LbSM) organise the storage as a circular log. They physically append modified/new data at the end (the logical head) of the log, which eliminates in-place updates and random writes. LbSM maintain a mapping of appended blocks and pages, they do not address issues related to version organisation such as additional write overhead introduced by the in-place invalidation.

We claim that the combination of a MV-DBMS using LbSM effectively addresses the characteristics of modern storage technologies. We further state that the most promising approach for append storage needs to be implemented *within the architecture and algorithms* of modern MV-DBMS. The following example offers a detailed description. Fig. 1 shows the invalidation process of different MV-DBMS (SI, SIAS), coupled to different types of storage managers ('in-place update' as original SI, page granularity LbMS, tuple granularity LbSM): three Transactions ($T1$, $T2$, $T3$) update data item X in serial order resulting in a relation that contains three different tuple versions of data item X. $T1$ creates the initial version X_0 of X. $T2$ issues the first update. In original SI, X_0 is invalidated in-place by setting its invalidation timestamp, subsequently X_1 is created. The update issued by $T3$ proceeds analogously and X_1 is invalidated in-place while X_2 is created as a new tuple version. Original SI, coupled to in-place update storage manager, writes X_0 and X_1 to the same location (random write) after the updates $T1$ and $T2$ (the initial page based on the free space). Original SI, coupled to a page append LbSM, will write X_0 and X_1 to pages with a higher page number (small sequential append). The payload of the write (updated versions/total versions per page) may be very low, yielding 'sparse' writes. Under SIAS X_0, X_1 and X_3 will be buffered, placed on the same page and appended altogether.

The *contributions* of this paper are as follows. We explore the performance implications of an approach combining LbSM and MV-DBMS optimised for Flash storage called SIAS (Snapshot Isolation Append Storage).

It organises versions in MV-DBMS as a simple backwards-linked list and assigns all versions of a data item a virtual ID. SIAS involves adapted algorithmic version invalidation handling and visibility rules. Furthermore, it is natively coupled to a LbSM and uses tuple granularity for logical append I/Os.

The *experimental results* under a TPC-C aligned OLTP load show that: a) 'page append LbSM' is up to 73% faster than traditional the 'in-place update' approach; b) using SIAS version-wise append we observe up to 2.99 times improvement in both IOPS and runtime; c) SIAS reduces the write overhead up to 52x; d) page-append LbSM yields equal write amount as the 'in-place update' approach; e) space reclamation due to deleted/invisible tuples is not suitable for append LbSMs in general and slows them down by approx. 40%; f) in SIAS using one local append region per relation is up to 5% faster than one global append region; g) using page remapping append with one global region is approx. 4.5% faster than using a local region; h) all append storage I/O performance scales with growing parallelism where in-place update approaches reach early saturation.

The paper is organised as follows: Section 2 provides a brief overview on related algorithmic approaches and systems; a general introduction of the used algorithms (SI and SIAS) is provided in Section 4; the main characteristics of Flash storage are summarised in Section 3. Section 2 describes combinations of in-place and append storage managers. Our experimental setup and framework are described in Section 6. The experimental results are discussed in Section 7.

2 Related Work

Snapshot Isolation (SI) is introduced and discussed in [2]. Specifics of a concrete SI implementation (PostgreSQL) are described in detail in [24,20]. As reported in [2] SI fails to enforce serializability. Recently a serializable version of SI was proposed [5] that is based on read/write dependency testing in serialization graphs. Serializable SI assumes that the storage provides enough random read throughput needed to determine the visible version of a tuple valid for a timestamp, making it ideal for Flash storage. [19] represents an alternative proposal for SI serializability. In addition serializable SI has been implemented in the new (but still unstable) version of PostgreSQL and will appear as a standard feature in the upcoming release.

SI [2] assumes a logical version organisation as a double-linked list and a two place invalidation, while making no assumption about the physical organisation. An improvement of SI called SI-CV, co-locating versions per transactions on pages has been proposed in [10].

Alternative approaches have been proposed in [7] and explored in [17,4] in combination with MVCC algorithms and special locking approaches. [17,4,7,11] explore a log/append-based storage manager. A performance comparison between different MVCC algorithms is presented in [6]. [15] offers insights to the implementation details of SI in Oracle and PostgreSQL. An alternative approach utilising transaction-based tuple collocation has been proposed in [10].

Similar chronological-chain version organisation has been proposed in the context of update intensive analytics [14]. In such systems data-item versions are never deleted, instead they are propagated to other levels of the memory hierarchy such as hard disks or Flash SSDs and archived. Any logical modification operation is physically realised as an append. SIAS on the other hand provides mechanisms to couple version visibility to (logical and physical) space management. Another difference is that SIAS uses transactional time (all timestamps are based on a transactional counter) as opposed to timestamps that correlate to logical time (dimension). Stonebraker et al. realised the concept of TimeTravel in PostgreSQL [22].

Multi-Version Database Systems. While Time-travel and MVCC approaches have been around for three decades, MV-DBMS approaches are nowadays applied in in-memory computing systems such as Hyper [13] or HYRISE [12] to handle mixed OLAP, OLTP loads, to handle database replication (Postgre-R) etc.

MV-DBMS are a good match for enterprise loads [14]. As discussed in [14], these are read-mostly; the percentage of writes is as low as approx. 17% (OLTP) and approx. 7% (OLAP) [14]. Since reads are never blocked under MVCC, in such settings there are clear performance benefits for the read-mostly enterprise workloads.

Multi-version approaches are widely spread in commercial and open source systems. Some MV-DBMS systems are: Berkeley DB (Oracle), IBM DB2, Ingres, Microsoft SQL Server 2005, Oracle, PostgreSQL, MySQL/InnoDB. And in addition *in-memory* systems such as Hyper [13], Hyder [3] etc.

Multi-Version approaches and MV-DBMS leverage the properties of new hardware. In this paper we investigate how these can be utilised to leverage I/O asymmetry of new storage technologies. Multi-version approaches can be used to leverage hardware characteristics of modern CPUs in transparently creating snapshots of in-memory pages [13] or to control data placement and caching in memory hierarchies.

Append Storage Management. LbSMs follow the principle of appending new data at the end of log structured storage. MV-DBMS alleviate appending of new data in principle, yet traditional approaches write data to arbitrary positions, updating data in-place or allocating new blocks. Such *traditional approaches*, implemented in current databases, address special properties of HDDs – especially their high sequential throughput and high latency access time on any type of I/O. They maintain clustering by performing in-place updates to optimise for read accesses, reducing the latency introduced by the properties of HDDs (rotational delay, positioning time). Thus implementations like SI in PostgreSQL rely on the in-place invalidation of old tuple versions. New storage technologies introduce fundamentally different properties (Section 3) and require optimised access methods. Especially low latency access time and fast random reads are not addressed yet and have to be leveraged.

LbSMs address the high throughput of large sequential writes on HDDs but destroy clustering, since new and updated data is not clustered with existing data yielding the same clustering attributes. Approaches using delta stores still require relatively expensive merge operations and generate overhead on read accesses [16].

The applicability of LbSMs for novel asymmetric storage has been partially addressed in [21,3] using page-granularity, whereas SIAS employs tuple-granularity (tuple append LbSM) much like the approach proposed in [4], which however invalidates tuples in-place. Given a page-append LbSM the invalidated page is remapped and persisted at the head of the log, hence no write-overhead reduction. In tuple-granularity, multiple new tuple-versions can be packed on a new page and written together.

3 Flash Memories

Enterprise Flash SSDs independent of their hardware interfaces (SATA, PCIe), exhibit significantly better performance and very different characteristics than traditional hard disks. Since most DBMS were build to compensate for the properties of HDDs, they tread SSDs as HDD replacement, which yields suboptimal performance. The most important characteristics of Flash are:

(i) Asymmetric read/write performance – reads are up to an order of magnitude faster than writes as a result of the physical NAND properties and their internal organisation. NAND memories introduce erase as an additional third operation together with read and write. Before performing a write, the whole block containing the page to be written must be erased. Writes should be evenly spread across the whole volume to avoid damage due to wear and increase endurance - wear-levelling. Hence no write in-place as on HDDs, instead copy-and-write. (ii) High random read performance (IOPS) – random reads for small block sites are up to hundred times faster than on an HDD. (iii) Low random write performance (IOPS) – small random writes are five to ten times slower than reads. Random writes depend one the fill-degree of device and incur a long term performance degradation due to Flash-internal fragmentation effects. (iv) Good sequential read/write transfer. Sequential operations are asymmetric, due to techniques as read ahead and write back caching the asymmetry is below 25%. (v) Suboptimal mixed load performance – Flash SSDs can handle pure loads (read or write) very well despite of the degree of randomness (random writes excluded). (vi) Parallelism – Compared to the typical hard drive and due to their multi-chip internal organisation Flash can handle much higher levels of I/O parallelism, [8],[1].

4 Algorithmic Description

In the following section we give a brief introduction to the SI algorithm as originally proposed in [2]. We then illustrate SI by using the implementation in PostgreSQL and point out differences and optimisations. Finally we present the SIAS algorithm.

4.1 Snapshot Isolation Introduction

SI is a timestamp based MVCC mechanism which assigns each running transaction exactly one timestamp and each data item two. The transaction's timestamp corresponds to the start of the transaction and the data item's timestamps correspond to the creation, respectively the invalidation of that item. An invalidation is issued on an update/deletion of an item. Each running transaction executes against its own version/snapshot of the committed state of the database. Isolated from effects of other concurrently running transactions, a transaction is allowed to read an older committed version of a data item instead of reading a newer, uncommitted version of the same item or being blocked/aborted. A snapshot describes the visible range of items the transaction is able to ”see” (facilitated by the timestamps). On an access to an item the transaction's timestamp is compared to the ones on the item. Items with a higher creation timestamp (inserted after the start of the transaction) are invisible and such with a lower (or equal) timestamp are visible to the transaction as long as they are committed and were not inserted concurrently. Reads are therefore never blocked by writes and changes made by a transaction are executed on its own snapshot which becomes visible to follow up transactions after its successful commit. Whether or not a commit is successful is determined at commit time by the transaction manager, which performs a write set comparison of the involved concurrent transactions. Overlapping write sets between concurrent transactions are not allowed and lead to the abort of at least one transaction since it is not allowed to have more than one update to an item. Two equivalent rules guarantee this behaviour: ”first-committer-wins” [2] and ”first-updater-wins” [2],[20]. The former corresponds to a deferred check at commit time, while the latter is enforced by immediate checks e.g. exclusive locks.

4.2 SIAS - Algorithm

Fig. 1 shows how different versions are handled under different approaches. SIAS [18] introduces a new addressing mechanism: (multiple) tuple versions are addressed as a chain by means of a virtual tuple ID (VID) that identifies the chain (as one logical item; all tuple versions in the chain share the same VID).

When a tuple-version is read the entry point is fetched first and the visibility can be determined for each VID. If the entry points timestamp is too high or equals a concurrent transaction, the predecessor version is fetched. Visibility information is coded within the algorithmic chain traversal access methods. Each version $n (n \neq 0)$ of a tuple is linked to its predecessor $n - 1$. The first version ($n = 0$) points to itself or uses a $NULL$ pointer. The VID identifies a chain; each member-tuple receives a unique tuple-ID (TID) as well as a *version count* that indicates its position within the chain. The newest member has the highest chain count and becomes the *entry point*. To speed up VID lookups an in-memory data structure, recording of all entry points is created (Sect. 4.3). The tuple structure used by SIAS is shown in Table 1 and illustrated in the following example. Assume two data items X and Y forming two chains; X was updated once and Y twice. The entry points are versions X_1 and Y_2 (marked bold in Table 1). Each version maintains a *pointer* to its predecessor forming a physical chain. The visibility is determined by a chain traversal, starting at the entry point applying

Table 1. SIAS - On-Tuple Information

Tuple	Creation Xmin	Predecessor Pointer	Predecessor Xmin (Xpred)	VID	Version Count
X0	15	X0	null	0x0	0
X1	38	X0	15	0x0	1
Y0	50	Y0	null	0x23	0
Y1	83	Y0	50	0x23	1
Y2	110	Y1	83	0x23	2

the SIAS algorithm rules – instead of reading an invalidation timestamp the creation timestamps of two subsequent versions are compared (xmin, xmin_pred).

SIAS verifies visibility of tuple versions based on the entry point, while SI inspects each tuple version individually. The number of chains equals the number of entry points (items) while the amount of tuple versions in the database can be much higher. SIAS entry-points represent a subset of all tuple-versions and at most one tuple-version per chain is visible to each transaction. The visibility control can discard a whole (sub-) chain of tuple-versions, depending on the value of the creation timestamp, thus saving I/O. Hence on average, SIAS has to read less tuple-versions to determine visibility, but may perform more read I/Os to fetch the appropriate version. The most recent version may not be the one visible for an older (longer running) transaction.

4.3 SIAS - Data Structures

SIAS introduces two data structures to organise the entry point information:

(i) $dstructI$: mapping of the VID to the chain member count.

(ii) $dstructII$: mapping of the VID to (the location of) the entry point (TID).

$dstructI$ accelerates verification of the following condition: is the tuple-version under inspection an entry-point or has the entry-point been changed by another committed transaction. This information can also be obtained by comparing the tuple ID (TID) of the read tuple version and the TID within $dstructII$, thus making $dstructI$ optional.

$dstructII$ is used to access the current entry-point.

The chaining mechanism based on VIDs has the following implications: (a) *The chain length* depends on whether old and invisible versions are retained/archived and on the update frequency and duration of transactions. The chain length is therefore defined by the total amount of updates on the respective tuple. (b) *The amount of extra reads* due to chain traversal depends on (c) *The amount of visible versions.*

5 Append Storage

In the following we briefly introduce our approaches to append storage in MV-DBMS. We classify the approaches in page-wise and tuple-wise LbSMs, further categorize them according to Figure 2 and explain them in more detail in the following sections.

5.1 Page-Append

The page-append LbSM describes append storage outside the database, without knowledge of the inertia of transmitted pages, implementing a 'holistic' page remapping paradigm. We utilise a default out of the box PostgreSQL running under a SI MVCC mechanism (Sect. 6), enhanced by LbSMs in the following variants:

SI-PG (SI – Page Global) denotes the traditional approach where pages are written to one single append region on the storage device; hence a global append region. It performs a remapping of page- and block addresses. We simulate variants with (SI-PG-V) and without a garbage collection mechanism (SI-PG-NV); we refer to SI-PG when describing both variants.

SI-PL (SI – Page Local) extends the SI-PG approach with multiple append regions. SI-PL receives more information about the content of the transmitted pages. SI-PL partitions the global append storage into multiple *local* append regions, dedicating each region to a single relation of the database. We simulate variants with (SI-PL-V) and without a garbage collection mechanism (SI-PL-NV); we refer to SI-PL when describing both variants.

PostgreSQL uses a space reclamation process called vacuum to garbage collect invisible versions (Sect. 5.4). SI-PG and SI-PL do not require changes to the MV-DBMS. They rather maintain a mapping of pages, performing block-address translation to generate *flash-aware* patterns. Both can be implemented as a layer between the device and the MV-DBMS. Although this already delivers some benefits for new storage media such as flash, our evaluation shows that those can be optimised by inherently integrating the storage manager into the MV-DBMS.

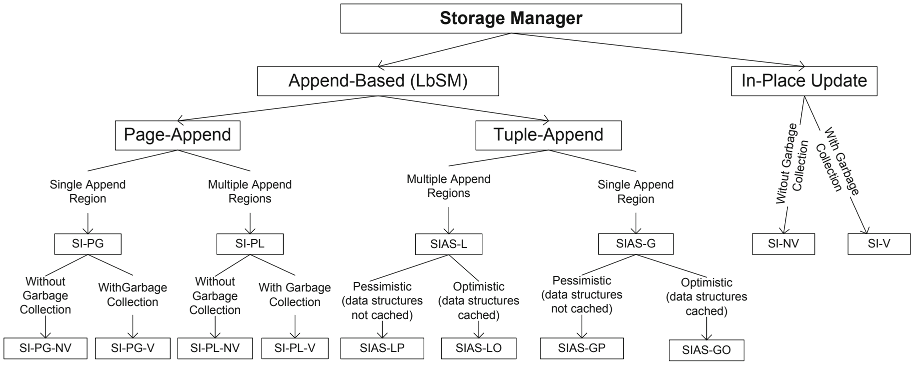


Fig. 2. Storage Manager Alternatives

5.2 SIAS - Tuple Append LbSM

We developed *SIAS (Snapshot Isolation Append Storage)* which algorithmically changes SI and improves on it by enabling tuple based appends without the need for in-place invalidation of old tuple versions. SIAS appends tuples to a page until it is filled and subsequently appends it to the head of the corresponding append log.

SIAS-L uses multiple append regions, where each region is dedicated to exactly one relation of the database. All pages in a local append region belong to the same relation and all tuples within a page belong to the same relation. Tuples are appended to a page of their relation, which is subsequently appended to the relation's local append region, after the page is filled completely or has reached a certain threshold.

SIAS-G uses one append region for all pages. Tuples within a single page belong to the same relation. Tuples get appended to a single page (analogously to SIAS-L) which is then appended to a global append region. The global append region maintains pages of all the relations of the MV-DBMS.

According to the SIAS concept we compare two variants of SIAS-L and SIAS-G, an *optimistic* approach which assumes that SIAS data structures are cached (SIAS-LO and SIAS-GO) and a *pessimistic* approach which fetches the data structures separately (SIAS-LP and SIAS-GP), thus resulting in four variants of SIAS. Since the test results of all SIAS variants showed the same performance independent of the garbage collection process, we omit the detailed report of these additional results in this paper and refer to SIAS-L and SIAS-G subsuming all four approaches.

5.3 In-Place - No Append

For the in-place approach we use the original Snapshot Isolation in two configurations:

SI-NV (SI No Vacuum) – deactivated garbage collection in PostgreSQL (vacuum),

SI-V (SI with Vacuum) – activated garbage collection (vacuum) in PostgreSQL.

5.4 Space Reclamation

In LbSMs written data is immutable, whereas in a MV-DBMS invalidated versions of tuples become invisible and occupy space which can be freed. The page-append LbSM has no knowledge about invalidated versions and therefore has to rely on methods within the MV-DBMS for space reclamation (e.g. vacuum in PostgreSQL).

Physical blocks get invalidated because the logical pages were remapped to another position and have to be physically deleted on the Flash device. The moment of execution is implementation dependent. On Flash an erase can only be performed in granularities of an erase unit - usually much larger than a page. Issuing an overwrite of a block (instead of deleting it) results in a remapping within the Flash device and therefore to unpredictable performance analogously to an in-place update (black box). Physical deletes should therefore only be issued in erase unit granularity (using trim). Pages which are still valid and reside within the unit which is about to be erased have to be re-mapped/re-inserted (append).

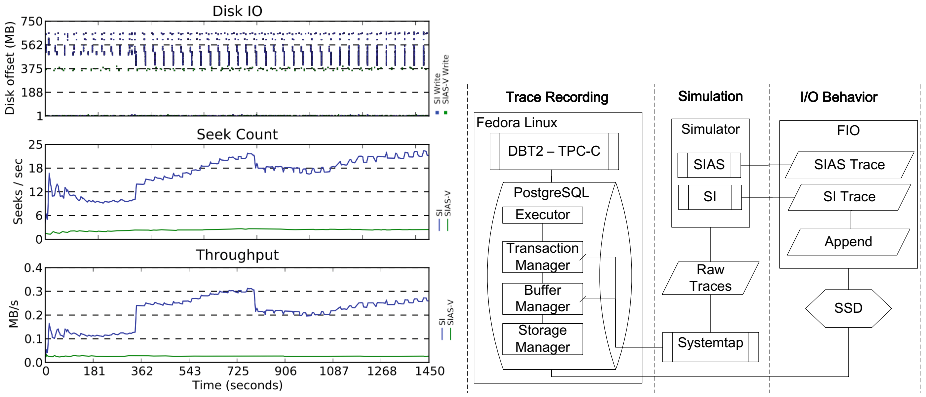
The tuple-append LbSM in SIAS is able to garbage collect single versions of a tuple. A page may contain still valid tuples which are simply re-inserted into the append log. Since each page is appended as soon as it is filled, the pages stay compact.

6 Evaluation

Our evaluation of the different LbSM alternatives (Sect. 2) is based upon a trace driven simulation, which we describe in the following paragraphs. We opted for simulation for two reasons: (a) to focus on the main characteristics of the multi versioning algorithms (i.e. exclude influences of the transaction-, storage- and buffer-manager as well as PostgreSQL's specific overhead); and (b) to compare a number of LbSM and SIAS alternatives. The simulator was validated against the PostgreSQL implementation of our baseline SIAS algorithm (see *validation* in this section). The simulation workload is created by an open source TPC-C implementation [23]. The simulation (Fig. 3(b)) comprises the following steps: (i) Recording of the raw-trace; (ii-a) Simulation of SIAS and SI resulting in I/O traces; (ii-b) remapping of the traces, creating SI-PL and SI-PG traces; (iii) I/O trace execution on a physical SSD (Intel X25-E SLC) using FIO; and (iv) validation using our SIAS prototype in PostgreSQL, which was installed on the same hardware. We describe all those steps in detail in the following paragraphs.

Instrumentation. A default, out of the box PostgreSQL (9.1.4.) was used to record the trace. It was instrumented utilising TPC-C (DBT2 v0.41)[23] with the PostgreSQL default page size of 8KB. All appends were conducted using this granularity. The used Fedora Linux (kernel 2.6.41) included the systemtap extension (translator 1.6; driver 0.152).

Raw Trace. The raw trace (Fig. 3(b)) contains: (i) tuples and the operations executed on them; (ii) the visibility decision for each tuple; (iii) the mapping of tuples to pages. We record each operation on a tuple and trace the visibility decision for that tuple. By setting probing points accordingly within the transaction- and buffer-manager, we eliminate their influence and are able to simulate the raw I/O of heap-data (non-index data) tuples. The resulting raw-trace is fed to the simulator.



(a) Blocktrace on SSD: SI-V vs. SIAS-LO (b) Simulation Process

Fig. 3. Blocktraces and Simulation Process

Simulator. SI and SIAS are simulated based on the raw trace including visibility checks and the resulting storage accesses. During the simulation the DB state is re-created according to the respective approach (Fig. 2). Hence the simulated databases always contain exactly the same tuples as the original DB. The only difference is the permutation of the tuples' location; tuples reside on different pages within the simulated DB.

SIAS creates a new tuple mapping: when a tuple is inserted into the original DB (raw trace), the tuple is inserted in its initial version, augmented by the SIAS attributes. The baseline SIAS algorithm (SIAS-L) algorithmically generates a local append, yielding one append region per relation. In order to simulate SIAS-G, an additional mapping is performed, analogous to the page-append LbSM.

As a result of the simulation process block-level traces are generated. These reflect the I/O access pattern that a DMBS would execute against the storage. Subsequently the block-level traces are executed on an a real SSD using FIO, which allows us to precisely investigate the influence of I/O parallelism and raw access.

FIO Trace. The FIO I/O benchmark guarantees controlled trace execution, repeatable results, configurable I/O parallelism and reliable metrics. FIO was configured using the *libaio* library accessing an Intel X25-E SSD via direct I/O as a raw device. The raw device had no filesystem layer in between. We consider the SSD as a black box, which means that no tuning for device-specific properties was applied. To avoid SSD state dependencies we executed a series of 8KB random writes after each single run.

Validation. We implemented the SIAS prototype in PostgreSQL. This prototype was validated under a TPC-C workload. The write patterns generated by our simulation and the PostgreSQL SIAS prototype are the same (see Fig. 3(a)). In terms of I/O parallelism both (PostgreSQL prototype and simulation) achieve: (i) compareable performance; (ii) similar write patterns; and (iii) the same net/gross write overhead reduction.

Load Characterisation. We used the DBT2 benchmark v0.41 [9] which is an open source TPC-C [23] implementation. DBT2 was configured for two traces. Both traces used 10 clients per warehouse and a total of 200 warehouses. *Trace I* with a runtime of 60 minutes and *Trace II* with a runtime of 90 minutes. Based on these two traces we also investigate the impact on space management, chain length etc.

7 Results

I/O Performance and Parallelism. We measured the performance of the algorithms discussed in Section 2 and shown in Fig. 2. We configured FIO with different queue depths (QD) ranging from 1 (no parallelism) to 32 (the maximum queue depth of our Intel X25-E SSD). *I/O performance:* In general, SI-V and SI-NV (SI with and without Vacuum/garbage collection) show the lowest performance for *Trace I* and *Trace II*: the read IOPS of SI-V are the lowest as depicted in Fig. 4a, 4b, 4c and 4d, therefore the runtime of SI-V and SI-NV is significantly higher (Fig. 4e and 4f). Figure 4a and 4c both illustrate the same trace. Figure 4a illustrates the differences between SIAS and SI-P in both global and local implementation variants. Figure 4c additionally displays the general in-place update approach of SI. Furthermore, the I/O performance (seeks, throughput, access distribution) over time for 32 QD is depicted in Fig. 6; SI-V needs more than twice the time of SIAS-L variants. The runtime of the page-append LbSM variants is 2.1x the runtime of SIAS for *Trace I* (Fig. 4d) and *Trace II* (Fig. 4e, 4f).

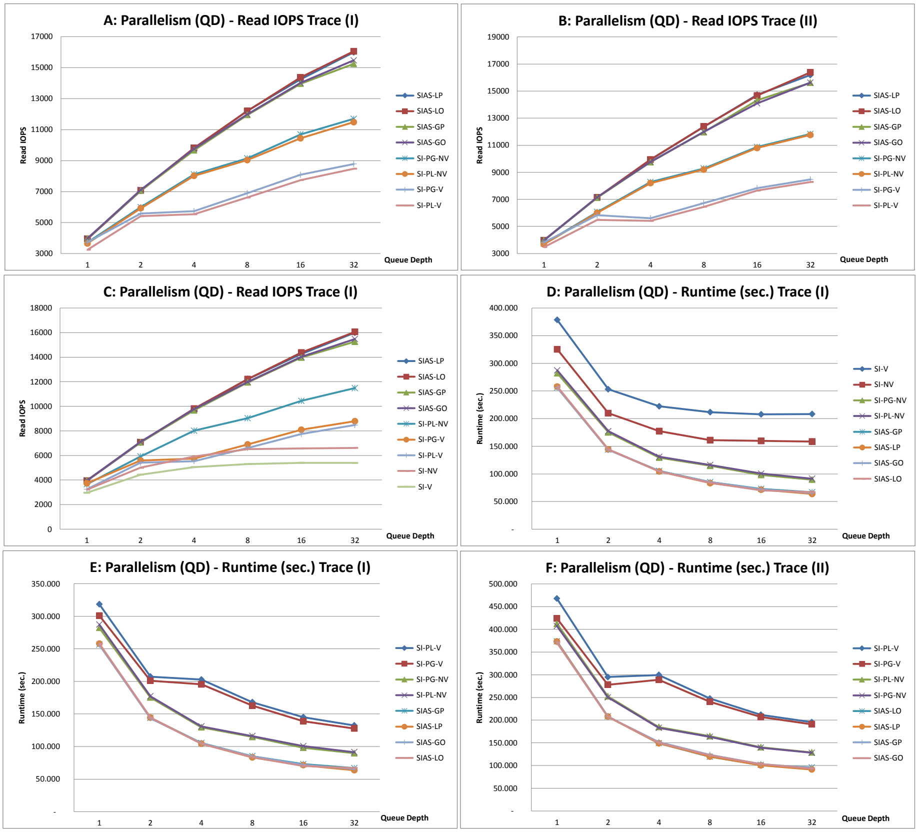


Fig. 4. I/O Parallelism: Read IOPS vs. Queue Depth for *Trace I* (a, c, d, e) and *Trace II* (b, f)

Without parallelism: (i) the SIAS I/O rate is 34% higher than SI-V and 23% higher than SI-NV; (ii) SI-NV is approx. 8% faster than SI-V.

I/O parallelism: Fig. 4 shows that SI-V and SI-NV improved up to a QD of two (2 parallel I/O requests), stagnated at four and reached saturation at a QD of eight; hence no leverage of parallelism. The page-append LbSM variants SI-PG and SI-PL are up to 73% faster than the in-place update SI variants SI-V and SI-NV (QD of 32). Without parallelism SI-PL is 13% faster than SI if garbage collection (Vacuum) is activated (up to 25% higher read IOPS than SI-V if vacuum is deactivated). SI-PL is marginally slower than SI-PG (Fig. 4c and 4d). Since SI-PL has to write at multiple locations, more seeks (random writes) are required than in SI-PG, as illustrated in Fig. 5 (Seek Count) – the append-log region for reads/writes of each relation is visible as a straight line.

With increasing parallelism approaches using one append region per relation have the advantage over single region approaches.

Garbage Collection (Vacuum): all variants with enabled vacuum are significantly slower than their counterparts. This trend is intensified by a higher degree of parallelism. In Fig. 4a and 4b we observe that vacuum creates significant overhead when using the page-append LbSM. Starting with a queue depth of four, page-append LbSM variants loose up to 35% IOPS when using vacuum (Fig. 4a and 4b). SI-PG-NV and SI-PL-NV scale up to the maximum queue depth experiencing a slight stagnation at queue depths larger than four (Fig. 4a and 4b). SI-PG-V and SI-PL-V benefit from higher queue depths but not as much as the variants with deactivated vacuum. Garbage collection mechanisms are therefore not beneficial for page-append LbSMs. SIAS scales almost linearly with increasing parallelism and benefits from a high queue depth. The difference between pessimistic and optimistic SIAS is not significant but enhances with increasing levels of parallelism as depicted overall in Fig. 4. Global and local variants of SIAS perform equally well at lower levels of parallelism. With increasing parallelism the local approach is approx. 5% faster than the global approach, hence making optimal use of the Flash device's parallelism. On *Trace I*, SIAS (in all variants) is up to 2.99x faster than SI-V, 2.43x faster than SI-PL-V/SI-PG-V and approx. 40% faster than SI-PG-NV/SI-PL-NV. Since the performance difference between the global and local implementation of SIAS is marginal and in favour of the local variant, it is not justified to create and maintain an additional page mapping as it is necessary for the global variant (SIAS-G). *Trace II* shows results analogous to *Trace I*. The I/O rate directly correlates with the runtime of the traces. The tendencies observed in this section are confirmed. The in-place approaches SI-V and SI-NV need the most time to complete the trace as depicted in Fig. 4d and Fig.6. SI-PL-NV and SI-PG-NV show almost identical runtime behaviour as well as SI-PL-V and SI-PG-V (Fig. 4e). SIAS is in all four implementations faster than the other approaches (Fig. 4).

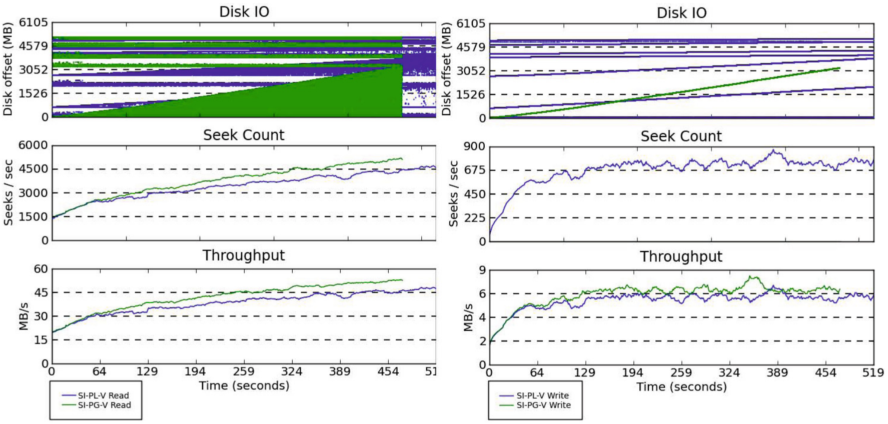


Fig. 5. Read Write Blocktrace on Physical Device: SI-PL vs. SI-PG

Read/Write Overhead. Non-Vacuum SI variants (SI-PG-NV, SI-PL-NV and SI-NV) write 966MB in *Trace I* and 1396MB in *Trace II*. SI variants performing Vacuum (SI-PG-V, SI-PL-V and SI-V) write 1304.6MB in *Trace I* and 1975.3 in *Trace II*. A key feature of SIAS is the significant write reduction of up to *52 times*. SIAS writes (in all variants) 25MB in *Trace I* and 39.9MB in *Trace II*. The write overhead is reduced to a fragment of the usual amount, which is a direct consequence of the out-of-place invalidation, logical tuple appends and dense filling of pages. The metadata update to invalidate a tuple version in SI leads to an update of the page in which this version resides, although the data-load of that version is unchanged. Additionally the new version has to be stored. SIAS avoids such metadata updates. Pages are packed more dense and tuple versions of subsequent access are most likely cached.

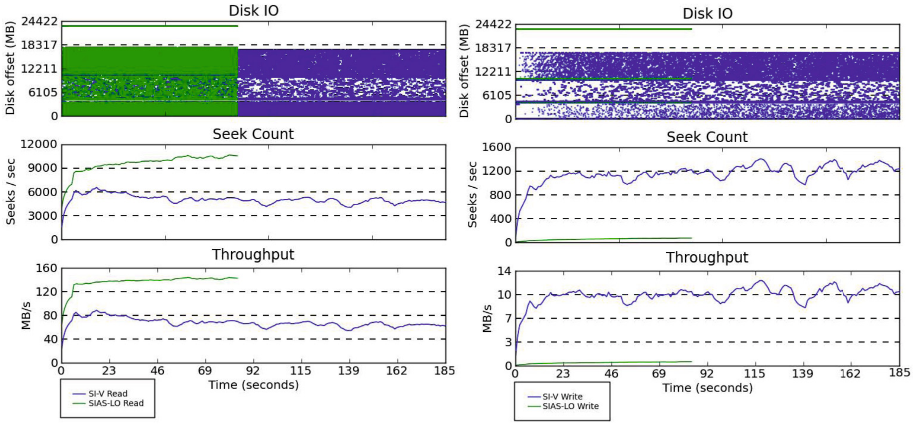


Fig. 6. Read Write Blocktrace on Physical Device: SI-V vs. SIAS-LO

8 Conclusion

We compared in-place storage management and page-/tuple-based LbSM approaches in conjunction with multi versioning databases on new storage technologies and elaborated the influence of one single or multiple append regions. Our findings show that while page-append LbSM approaches are better suitable for new storage technologies, they can be optimised by implementing tuple-based LbSM directly into the MV-DBMS. We implemented SIAS, a tuple-append LbSM within a MV-DBMS which algorithmically generates local append behaviour. SIAS leverages the properties of Flash storage, achieves high performance, scales almost linearly with growing parallelism and exhibits a significant write reduction. Our experimens show that: a) traditional LbSM approaches are up to 73% faster than their in-place update counterparts; b) SIAS tuple-version granularity append is up to 2.99x faster (IOPS and runtime) than in-place update approaches; c) SIAS reduces the write overhead up to 52 times; d) in SIAS using exclusive append regions per relation is up to 5% faster than using one append region for all relations.

Acknowledgements. This work was supported by the DFG (Deutsche Forschungsgemeinschaft) project “Flashy-DB”.

References

1. Agrawal, N., Prabhakaran, V., et al.: Design tradeoffs for ssd performance. In: Proc. ATC 2008, pp. 57–70 (2008)
2. Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E., O'Neil, P.: A critique of ansi sql isolation levels. In: Proc. SIGMOD 1995, pp. 1–10 (1995)
3. Bernstein, P.A., Reid, C.W., Das, S.: Hyder - a transactional record manager for shared flash. In: CIDR 2011, pp. 9–20 (2011)
4. Bober, P., Carey, M.: On mixing queries and transactions via multiversion locking. In: Proc. IEEE CS Intl. Conf. No. 8 on Data Engineering, Tempe, AZ (February 1992)
5. Cahill, M.J., Röhm, U., Fekete, A.D.: Serializable isolation for snapshot databases. In: Proc. SIGMOD 2008, pp. 729–738 (2008)
6. Carey, M.J., Muhanna, W.A.: The performance of multiversion concurrency control algorithms. ACM Trans. on Computer Sys. 4(4), 338 (1986)
7. Chan, A., Fox, S., Lin, W.-T.K., Nori, A., Ries, D.R.: The implementation of an integrated concurrency control and recovery scheme. In: Proc. SIGMOD 1982 (June 1982)
8. Chen, F., Koufaty, D.A., Zhang, X.: Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In: Proc. SIGMETRICS 2009 (2009)
9. Database Test Suite DBT2, `http://osdldbt.sourceforge.net`
10. Gottstein, R., Petrov, I., Buchmann, A.: SI-CV: Snapshot isolation with co-located versions. In: Nambiar, R., Poess, M. (eds.) TPCTC 2011. LNCS, vol. 7144, pp. 123–136. Springer, Heidelberg (2012)
11. Gottstein, R., Petrov, I., Buchmann, A.: Aspects of append-based database storage management on flash memories. In: Proc. of DBKDA 2013, pp. 116–120. IARIA (2013)
12. Grund, M., Krüger, J., Plattner, H., Zeier, A., Cudre-Mauroux, P., Madden, S.: Hyrise: a main memory hybrid storage engine. Proc. VLDB Endow. 4(2), 105–116 (2010)
13. Kemper, A., Neumann, T.: Hyper: A hybrid oltp and olap main memory database system based on virtual memory snapshots. In: ICDE (2011)
14. Krueger, J., Kim, C., Grund, M., Satish, N., Schwalb, D., Chhugani, J., Plattner, H., Dubey, P., Zeier, A.: Fast updates on read-optimized databases using multi-core cpus. Proc. VLDB Endow. 5(1), 61–72 (2011)
15. Majumdar, D.: A quick survey of multiversion concurrency algorithms
16. O'Neil, P., Cheng, E., Gawlick, D., O'Neil, E.: The log-structured merge-tree (1996)
17. Petrov, I., Gottstein, R., Ivanov, T., Bausch, D., Buchmann, A.P.: Page size selection for OLTP databases on SSD storage. JIDM 2(1), 11–18 (2011)
18. R. Gottstein, I. Petrov and A. Buchmann. SIAS: On Linking Multiple Tuple Versions in Append DBMS (submitted)
19. Revilak, S., O'Neil, P., O'Neil, E.: Precisely serializable snapshot isolation (pssi). In: Proc. ICDE 2011, pp. 482–493 (2011)
20. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database Systems Concepts, 4th edn. McGraw-Hill Higher Education (2001)
21. Stoica, R., Athanassoulis, M., Johnson, R., Ailamaki, A.: Evaluating and repairing write performance on flash devices. In: Boncz, P.A., Ross, K.A. (eds.) Proc. DaMoN 2009, pp. 9–14 (2009)
22. Stonebraker, M., Rowe, L.A., Hirohama, M.: The implementation of postgres. IEEE Trans. on Knowledge and Data Eng. 2(1), 125 (1990)
23. TPC Benchmark C Standard Specification, `http://www.tpc.org/tpcc/spec/tpcc_current.pdf`
24. Wu, S., Kemme, B.: Postgres-r(si): Combining replica control with concurrency control based on snapshot isolation. In: Proc. ICDE 2005, pp. 422–433 (2005)

Lossless Horizontal Decomposition with Domain Constraints on Interpreted Attributes

Ingo Feinerer[1], Enrico Franconi[2], and Paolo Guagliardo[2]

[1] Vienna University of Technology
ingo.feinerer@tuwien.ac.at
[2] Free University of Bozen-Bolzano
{franconi,guagliardo}@inf.unibz.it

Abstract. *Horizontal decomposition* is the process of splitting a relation into sub-relations, called *fragments*, each containing a subset of the rows of the original relation. In this paper, we consider horizontal decomposition in a setting where some of the attributes in the database schema are *interpreted* over a specific domain, on which a set of special predicates and functions is defined.

We study the *losslessness* of horizontal decomposition, that is, whether the original relation can be reconstructed from the fragments by union, in the presence of integrity constraints on the database schema. We introduce the new class of *conditional domain constraints* (CDCs), restricting the values the interpreted attributes may take whenever a certain condition holds on the non-interpreted ones, and investigate lossless horizontal decomposition under CDCs in isolation, as well as in combination with functional and unary inclusion dependencies.

1 Introduction

Horizontal decomposition (or *fragmentation*) is the process of splitting a relation into sub-relations on the same attributes and of the same arity, called *fragments*, each containing a subset of the rows of the original relation. Fragmentation (horizontal and/or vertical) plays an important role in distributed database systems, where fragments of a relation are scattered over several (local or remote) sites, as it typically increases the system's throughput by permitting a number of transactions to execute concurrently and by allowing the parallel execution of a single query as a set of subqueries that operate on fragments [11]. This is especially crucial in data intensive applications. Horizontal decomposition is also important for improving schema design, in that it can alleviate the problem of dependencies not being preserved, that might arise in the conversion into Boyce-Codd normal form (BCNF) when a non-key set of attributes determines part of the key [7].

The study of horizontal decomposition in the literature [2,4,5,3,7,10] focused on uninterpreted data, that is, settings where data values can only be compared for equality. However, most real-world applications make use of data values belonging to domains with a richer structure (e.g., ordering) on which a variety of other restrictions besides equality can be expressed (e.g., being within a range or

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 77–91, 2013.

above a threshold). Examples are dimensions, weights and prices in the database of a postal service or shipping company, or the various amounts (credits, debits, exchange and interest rates, etc.) recorded in a banking application. It is therefore of practical interest to consider a scenario where some of the attributes in the database schema are *interpreted* over a specific domain, such as the reals or the integers, on which a set of predicates (e.g., smaller/greater than) and functions (e.g., addition and subtraction) are defined, according to a first-order theory $\mathfrak{C}$. In this paper, we consider horizontal decomposition in such a setting, where fragments are defined by means of selection queries with equalities and inequalities on the non-interpreted attributes, extended with constraints on the interpreted attributes expressed by formulae of $\mathfrak{C}$. As it is customary in the study of database decompositions, we make the pure universal relation assumption (URA) [1], that is, we limit our investigation to a database schema consisting of only one relation symbol. This simplifying assumption, even though it might seem unrealistic in general, is nevertheless satisfied by a surprising number of real-world databases.

We study the *losslessness* of horizontal decomposition, that is, whether the original relation can be reconstructed from the fragments by union, in the presence of integrity constraints on the database schema. Specifically, we investigate the problem of deciding whether a set of user views specified by selection queries (on interpreted attributes), which form a horizontal decomposition, is lossless under constraints. This is relevant to applications in data privacy (where losslessness is undesirable because it means that the views expose the whole database) and in view updates (where losslessness is desirable in that it allows changes on the views to be univocally propagated to the underlying database).

We introduce the new class of *conditional domain constraints* (CDCs) which by means of a $\mathfrak{C}$-formula restrict the values that the interpreted attributes can take whenever certain conditions hold on the non-interpreted ones. We speak of $\mathfrak{C}$-CDCs to emphasise that the restriction on the interpreted attributes is given by a formula in $\mathfrak{C}$. Depending on the expressive power of $\mathfrak{C}$, CDCs can capture constraints naturally arising in practise, such as, in the above mentioned postal service scenario, that a parcel of type "letter" not marked as "oversized" weights less than 20 grammes and each of its dimensions (height, width, depth) is less than 30 centimetres. We do not commit to any specific language $\mathfrak{C}$ and we simply assume that $\mathfrak{C}$ is closed under negation.

First of all, we show how to check whether a set of $\mathfrak{C}$-CDCs is consistent by means of a characterisation in terms of satisfiability in $\mathfrak{C}$, which directly yields a decision procedure whenever the latter is decidable. This is the case, e.g., for the so-called *Unit Two Variable Per Inequality* (UTVPI) constraints – a fragment of linear arithmetic over the integers whose formulae, which we refer to as UTVPIs, have at most two variables and variables have unit coefficients – as well as for boolean combinations of UTVPIs. We prove that deciding consistency is in NP for the former language, and is NP-complete for the latter.

Then, we give a characterisation of lossless horizontal decomposition under $\mathfrak{C}$-CDCs in terms of unsatisfiability in $\mathfrak{C}$, which again directly yields a decision procedure whenever the latter is decidable, and we prove that deciding losslessness is in co-NP for the language of UTVPIs, and it is co-NP-complete for the language consisting of boolean combinations of UTVPIs.

Finally, we study lossless horizontal decomposition under CDCs and traditional integrity constraints, and show that functional dependencies (FDs) can be allowed without any restriction, as they do not interact with CDCs, whereas this is not the case for unary inclusion dependencies (UINDs). We provide a *domain propagation rule* that, under certain restrictions on the CDCs, fully captures the interaction between CDCs and UINDs (on both interpreted and non-interpreted attributes) w.r.t. the losslessness of horizontal decomposition, which allows to "separate" the UINDs from the CDCs and employ the general technique in this case as well.

Related Work. De Bra [2,4,5] developed a theory of horizontal decomposition to partition a relation into two sub-relations, one of which satisfies a set of specified FDs, while the other does not. The approach is based on constraints that capture empirically observed "implications" between sets of FDs and "exceptions" to sets of FDs, for which a sound and complete set of inference rules is provided. Maier and Ullman [10] study how to propagate insertions and deletions, and how to answer selection queries, in a setting with physical and virtual fragments, where the losslessness of the decomposition is taken as an underlying assumption. Ceri et al. [3] investigate the problem of finding an optimal horizontal partitioning w.r.t. the number of accesses to different portions of data.

Structure of the Paper. The rest of the paper is organised as follows: in Sec. 2 we formally define the notion of horizontal decomposition and introduce the class of CDCs; in Sec. 3 we show how to check for the consistency of a set of CDCs; the study of lossless horizontal decomposition under CDCs is carried out in Sec. 4, and it is then extended to CDCs in combination with FDs and UINDs in Sec. 5; we conclude in Sec. 6 with a brief discussion of the results and by pointing out future research directions.

2 Preliminaries

We start by introducing notation and basic notions that will be used throughout the paper. We first provide some general definitions, which we then specialise to the case we will investigate in the upcoming sections.

Basics. A (*database*) *schema* (also called a *relational signature*) is a finite set $\mathbf{S}$ of relation symbols, where each $S \in \mathbf{S}$ has arity $|S|$ and its positions are partitioned into *interpreted* and *non-interpreted* ones. Let **dom** be a possibly infinite set of arbitrary values, and let **idom** be a set of values from a specific domain (such as the integers $\mathbb{Z}$) on which a set of predicates (e.g., $\leq$) and functions (e.g., $+$) are defined, according to a first-order language $\mathfrak{C}$ closed under negation. An *instance* over a schema $\mathbf{S}$ associates each $S \in \mathbf{S}$ with a relation S^I of appropriate arity on $\mathbf{dom} \cup \mathbf{idom}$, called the *extension* of S under I, such that the values for the interpreted and non-interpreted positions of S are taken from **idom** and **dom**, respectively. The set of elements of $\mathbf{dom} \cup \mathbf{idom}$ that occur in an instance I is the *active domain* of I, denoted by $\mathbf{adom}(I)$. An instance is finite if its active domain is, and all instances are assumed to be finite unless otherwise specified.

Constraints. Let $\mathfrak{L}$ be a domain-independent (that is, safe-range) fragment of first-order logic over a relational signature $\mathbf{S}$ with constants $\mathbf{dom} \cup \mathbf{idom}$ under the standard name assumption, extended with predicates and functions interpreted on $\mathbf{idom}$ according to $\mathfrak{C}$. A *constraint* is a closed formula in $\mathfrak{L}$. The sets of relation symbols (from $\mathbf{S}$) and constants occurring in a constraint φ are denoted by $\mathsf{sig}(\varphi)$ and $\mathsf{const}(\varphi)$, respectively. We extend $\mathsf{sig}(\cdot)$ and $\mathsf{const}(\cdot)$ to sets of constraints in the natural way. For a set of constraints Γ, we say that an instance I over $\mathsf{sig}(\Gamma)$ is a *model* of (or *satisfies*) Γ, and write $I \models \Gamma$, to indicate that the relational structure $\mathcal{I} = \langle \mathbf{adom}(I) \cup \mathsf{const}(\Gamma), I \rangle$ makes every formula φ in Γ true w.r.t. the semantics of $\mathfrak{L}$. We write $I \models \varphi$ to indicate that I satisfies φ. A set of constraints Γ *entails* (or *logically implies*) a constraint φ, written $\Gamma \models \varphi$, if every finite model of Γ also satisfies φ. All sets of constraints in this paper are assumed to be finite.

We consider a *source schema* $\mathbf{R}$, consisting of a single relation symbol R, and a *decomposed schema* $\mathbf{V}$, disjoint with $\mathbf{R}$, of *view* symbols with the same arity as R. We formally define horizontal decompositions as follows.

Definition 1 (Horizontal Decomposition). *Let* $\mathbf{V} = \{V_1, \dots, V_n\}$ *and let* $\mathbf{R} = \{R\}$. *Let* Δ *be a set of constraints over* $\mathbf{R}$, *and let* Σ *be a set of exact view definitions, one for each* $V_i \in \mathbf{V}$, *of the form* $\forall \overline{x} \,.\, V_i(\overline{x}) \leftrightarrow \varphi(\overline{x})$, *where* φ *is a safe-range formula over* $\mathbf{R}$. *We say that* Σ *is a* horizontal decomposition of $\mathbf{R}$ into $\mathbf{V}$ under Δ *if* $\Delta \cup \Sigma \models \forall \overline{x} \,.\, V_i(\overline{x}) \rightarrow R(\overline{x})$ *for every* $V_i \in \mathbf{V}$. *Moreover,* Σ *is* lossless *if* $\Delta \cup \Sigma \models \forall \overline{x} \,.\, R(\overline{x}) \leftrightarrow V_1(\overline{x}) \vee \dots \vee V_n(\overline{x})$.

For the sake of simplicity, we assume w.l.o.g. that the first k positions of R and of each $V \in \mathbf{V}$ are non-interpreted, while the remaining ones are interpreted. Under this assumption, instances over $\mathbf{R} \cup \mathbf{V}$ associate each symbol with a subset of $\mathbf{dom}^k \times \mathbf{idom}^{|R|-k}$. Throughout the paper, we further assume that a variable appearing in the i-th position of R is named x_i if $i \leq k$, and y_{i-k} otherwise. Clearly, this is also w.l.o.g. as it can be easily achieved by renaming.

For what concerns integrity constraints on the source schema, we introduce the class of *conditional domain constraints* (CDCs), which restrict the admissible values at interpreted positions by means of a formula in $\mathfrak{C}$, whenever a condition holds on the non-interpreted ones. Such constraints have the form

$$\forall \overline{x}, \overline{y} \,.\, \big(R(\overline{x}, \overline{y}) \wedge \overline{x}' = \overline{a} \wedge \overline{x}'' \neq \overline{b} \big) \rightarrow \delta(\overline{y}) \ , \tag{1}$$

where $\overline{x}'$ and $\overline{x}''$ consist of variables from $\overline{x}$, possibly with repetitions, and $\delta(\overline{y}) \in \mathfrak{C}$. To explicitly indicate that the consequent of a CDC is a formula in $\mathfrak{C}$, we refer to it as a $\mathfrak{C}$-CDC. For simplicity of notation, we omit the universal quantifier and the R-atom from (1).

We consider horizontal decompositions where the symbols in the decomposed schema are defined by means of selection queries with equalities and inequalities between variables and constants at non-interpreted positions, extended with conditions in $\mathfrak{C}$ on the interpreted positions. Formally, each $V \in \mathbf{V}$ is defined by a formula of the form

$$\forall \overline{x}, \overline{y} \,.\, V(\overline{x}, \overline{y}) \leftrightarrow \big(R(\overline{x}, \overline{y}) \land \overline{x}' = \overline{a} \land \overline{x}'' \neq \overline{b} \land \sigma(\overline{y})\big) \ , \tag{2}$$

with $\overline{x}'$ and $\overline{x}''$ consisting of variables from $\overline{x}$, possibly with repetitions, and $\sigma(\overline{y}) \in \mathfrak{C}$. In the following, we will simply write (2) as $V\colon \overline{x}' = \overline{a} \land \overline{x}'' \neq \overline{b} \land \sigma(\overline{y})$.

The class of *Unit Two Variable Per Inequality* constraints (UTVPIs), a.k.a. *generalised 2SAT* (G2SAT) constraints, is a fragment of linear arithmetic constraints over the integers. UTVPIs have the form $ax + by \leq d$, where x and y are integer variables, $a, b \in \{-1, 0, 1\}$ and $d \in \mathbb{Z}$. As the integer domain allows to represent real numbers with fixed precision as well, this class of constraints is suitable for most applications. Note that $ax + by \geq d$ is equivalent to $a'x + b'y \leq -d$, with $a' = -a$ and $b' = -b$, and that $ax + by < d$ and $ax + by > d$ can be rewritten as $ax + by \leq d'$ and $ax + by \geq d''$, respectively, with $d' = d - 1$ and $d'' = d + 1$. Thus, UTVPIs are powerful enough not only to express comparisons ($\leq$, $<$, $>$, $\geq$) between two variables and between a variable and an integer, but also to compare the sum or difference of two variables with an integer. Observe that, even though equality is not directly expressible within a single UTVPI, the equality $x = y$ in the consequent of a CDC, where y is a variable or an integer, can be expressed by means of two CDCs with the same antecedent, where one has $x \leq y$ as consequent, the other $x \geq y$. Equality between the sum or difference of two variables and an integer in a CDC is expressed in a similar way.

We denote by UTVPI the language of UTVPIs and by bUTVPI the language consisting of boolean combinations of UTVPIs, in turn called bUTVPIs. Checking whether a set of UTVPIs is satisfiable can be done in polynomial time [12,9,8], and the satisfiability problem for bUTVPI is NP-complete [13].

We conclude the preliminaries by introducing an example based on UTVPI-CDCs, which we will then use and further extend in the upcoming sections.

Example 1. Let R be on Name, Department, Position, Salary and Bonus, in this order, where the last two are interpreted over the integers, let $a =$ "ICT" and $b =$ "Manager", and let Δ consist of the following UTVPI-CDCs:

$$x_2 = a \ \rightarrow \ y_1 + y_2 \leq 5 \ ; \qquad x_3 = b \ \rightarrow \ y_2 \geq 2 \ ; \qquad \top \rightarrow \ y_1 - y_2 \geq 0 \ .$$

Intuitively, the above constraints state that employees working in the ICT department have a total income (salary plus bonus) of at most 5 – say – thousands of euros per month; that employees who work as managers receive a bonus of at least 2, and that employees never get a bonus greater than their salary.

3 Consistency of Conditional Domain Constraints

Before turning our attention to horizontal decomposition, we first deal with the relevant problem of determining whether a set of generic $\mathfrak{C}$-CDCs is consistent. Indeed, it is important to make sure that the integrity constraints on the source schema are consistent, as any horizontal decomposition would be meaninglessly lossless otherwise. In this section, we will characterise the consistency of a set of $\mathfrak{C}$-CDCs in terms of satisfiability in $\mathfrak{C}$. We will show that deciding consistency is in NP when $\mathfrak{C}$ is the language UTVPI, and is NP-complete when $\mathfrak{C}$ is the language bUTVPI. Moreover, the technique employed here will serve as the basis for the approach we follow in Sec. 4 in the study of lossless horizontal decomposition.

Since CDCs are universally-quantified implicational formulae, any set thereof is always trivially satisfied by the empty instance, thus we say that a set of CDCs is consistent if it has a non-empty model. Observe that, given their form, CDCs affect only one tuple at a time, hence whether an instance satisfies a set of CDCs depends on each tuple of the instance in isolation from the others. Indeed, a set of CDCs is satisfiable if and only if it is satisfiable on an instance consisting of only one tuple, and so we can restrict our attention to single tuples. But, as **dom** and **idom** are infinite domains in general, we cannot simply guess a tuple of values and check that it satisfies the CDCs. However, it is not necessary to know which values a tuple actually contains at non-interpreted positions, but only whether such values satisfy the equalities and inequalities in the antecedent of each CDC. To this end, with each equality or inequality between a variable x_i and a constant a we associate a propositional variable p_i^a, whose truth-value indicates whether the value in the i-th position is a. To each valuation of such variables corresponds the (possibly infinite) set of tuples that satisfy the conditions specified by the names of the variables. For instance, a valuation of $\{p_1^a, p_2^b\}$ that assigns true to p_1^a and false to p_2^b identifies all the tuples in which the value of the first element is a and the value of the second is different from b. Thus, for each such valuation it is possible to determine which equalities and inequalities are satisfied. Some care is only needed with valuations of propositional variables that refer to the same position (i.e., with the same subscript) but to different constants (i.e., with different superscripts). For example, p_1^a and p_1^b (with $a \neq b$) should never be both evaluated to true.

Roughly said, checking whether a set of $\mathfrak{C}$-CDCs Δ is consistent amounts to, as we shall see, first building from Δ a propositional theory where equalities and inequalities are replaced by propositional variables as above, and then finding a valuation α in which two propositional variables that refer to the same position but to different constants are not both evaluated to true, and for which the set of $\mathfrak{C}$-formulae that apply under α (obtained by "filtering" the propositional theory with α) is satisfiable.

For each $\mathfrak{C}$-CDC $\varphi \in \Delta$, which we recall has the form (1), we construct the propositional formula

$$\mathsf{prop}(\varphi) = \top \wedge \Big[\bigwedge_{\substack{x_i \in \mathsf{var}(\overline{x}') \\ x_i = \overline{x}'[j]}} p_i^{\overline{a}[j]} \Big] \wedge \Big[\bigwedge_{\substack{x_i \in \mathsf{var}(\overline{x}'') \\ x_i = \overline{x}''[j]}} \neg p_i^{\overline{b}[j]} \Big] \to v \ , \tag{3}$$

where v is a fresh propositional variable associated with the $\mathfrak{C}$-formula $\delta(\overline{y})$ in the consequent of φ. We call $\varPi_\Delta = \{ \mathsf{prop}(\varphi) \mid \varphi \in \Delta \}$ the *propositional theory associated with* Δ. Each propositional formula in $\varPi_\Delta$ is of the form $\top \wedge L \to v$, with L a (possibly empty) conjunction of literals and v a propositional variable associated with a $\mathfrak{C}$-constraint, denoted by $\mathsf{constr}(v)$. We omit $\top$ in the l.h.s. of the implication if L is non-empty, and consider the set $\mathsf{var}(\Pi_\Delta)$ of propositional variables occurring in Π_Δ partitioned into $\mathsf{pvar}(\Pi_\Delta) = \{\, \mathsf{var}(L) \mid (L \to v) \in \Pi_\Delta \,\}$ and $\mathsf{cvar}(\Pi_\Delta) = \mathsf{var}(\Pi_\Delta) \setminus \mathsf{pvar}(\Pi_\Delta)$. For a propositional theory Π_Δ, we build the following *auxiliary theory*:

$$\Pi_\bot = \{\, p_i^a \wedge p_i^b \to \bot \mid a \neq b,\ p_i^a, p_i^b \in \mathsf{pvar}(\Pi_\Delta) \,\} \ , \tag{4}$$

intuitively stating that distinct values are not allowed in the same position.

Example 2. The propositional theory associated with Δ from Example 1 is $\Pi_\Delta = \{\, p_2^a \to v_1,\ p_3^b \to v_2,\ \top \to v_3 \,\}$, with $\mathsf{pvar}(\Pi_\Delta) = \{\, p_2^a, p_3^b \,\}$ and $\mathsf{cvar}(\Pi_\Delta) = \{\, v_1, v_2, v_3 \,\}$. The corresponding auxiliary theory is $\Pi_\bot = \varnothing$. The association constr between the propositional variables in $\mathsf{cvar}(\Pi_\Delta)$ and UTVPI-formulae is given by $\{\, v_1 \mapsto y_1 + y_2 \leq 5,\ v_2 \mapsto y_2 \geq 2,\ v_3 \mapsto y_1 - y_2 \geq 0 \,\}$.

For a valuation α of $\mathsf{pvar}(\Pi_\Delta)$, from Π_Δ we build the set

$$\Pi_\Delta^\alpha = \{\, \mathsf{constr}(v) \mid (\top \wedge L \to v) \in \Pi_\Delta,\ L \text{ is empty or } \alpha(L) = \mathtt{T} \,\} \ , \tag{5}$$

consisting of $\mathfrak{C}$-constraints associated with propositional variables that occur in some formula of Π_Δ whose l.h.s. holds true under α. We call Π_Δ^α the *α-filtering of* Π_Δ. Note that, since $\mathfrak{C}$ is closed under negation, all the constraints in Π_Δ^α are $\mathfrak{C}$-formulae.

We can now state the main result of this section, characterising the consistency of a set of $\mathfrak{C}$-CDCs in terms of satisfiability in $\mathfrak{C}$. We remark that the result holds in general for any language $\mathfrak{C}$, not necessarily closed under negation. This requirement will however become essential in the upcoming Sec. 4 and Sec. 5.

Theorem 1. *Let Δ be a set of $\mathfrak{C}$-CDCs, Π_Δ be the propositional theory associated with Δ, and $\Pi_\bot$ be the corresponding auxiliary theory. Then, Δ is consistent if and only if there exists a valuation α of* $\mathsf{pvar}(\Pi_\Delta)$ *satisfying $\Pi_\bot$ such that Π_Δ^α is satisfiable.*

Clearly, whenever the satisfiability of sets of $\mathfrak{C}$-formulae is decidable, Theorem 1 directly yields an algorithm to check whether a set of $\mathfrak{C}$-CDCs is consistent. We illustrate this in our running example with UTVPI-CDCs.

Example 3. With respect to Π_Δ of Example 2, consider the valuation $\alpha = \{\, p_2^a \mapsto \mathtt{T},\ p_3^b \mapsto \mathtt{F} \,\}$, for which we have $\Pi_\Delta^\alpha = \{\, y_1 + y_2 \leq 5,\ y_1 - y_2 \geq 0 \,\}$. Obviously, α satisfies the (empty) auxiliary theory $\Pi_\bot$ for Π_Δ. In addition, Π_Δ^α is satisfiable, as for instance $\{\, y_1 \mapsto 3,\ y_2 \mapsto 2 \,\}$ is a solution to every UTVPI in it.

The *consistency problem* for $\mathfrak{C}$-CDCs is the problem that takes as input a set of $\mathfrak{C}$-CDCs Δ and answers the question: "Is Δ consistent?" In light of Theorem 1, the problem of checking whether a given set of $\mathfrak{C}$-formulae is satisfiable reduces to checking consistency for $\mathfrak{C}$-CDCs. Indeed, every instance of the satisfiability problem for $\mathfrak{C}$, asking whether a set F of $\mathfrak{C}$-formulae is satisfiable, can be encoded into an instance of the consistency problem for $\mathfrak{C}$-CDCs by considering, for each $\mathfrak{C}$-formula in F, a $\mathfrak{C}$-CDC which has that formula in the consequent and $\top$ in the antecedent (i.e., no equalities and inequalities). Thus, whenever the satisfiability problem in $\mathfrak{C}$ is C-hard, for some complexity class C, then the consistency problem for $\mathfrak{C}$-CDCs is also C-hard. We have the following complexity results concerning the consistency problem for UTVPI-CDCs and bUTVPI-CDCs.

Theorem 2. *The consistency problem for $\mathfrak{C}$-CDCs is in NP when $\mathfrak{C} = \mathsf{UTVPI}$, and is NP-complete when $\mathfrak{C} = \mathsf{bUTVPI}$.*

4 Losslessness of Horizontal Decomposition

The technique described in Sec. 3 can be extended to, and applied for, checking whether a horizontal decomposition is lossless under $\mathfrak{C}$-CDCs. In this section, given a horizontal decomposition Σ of R into $V_1, \dots, V_n$ and a set Δ of $\mathfrak{C}$-CDCs, we will characterise the losslessness of Σ under Δ in terms of unsatisfiability in $\mathfrak{C}$. Moreover, we will also show that deciding losslessness is in $\mathsf{co\text{-}NP}$ when $\mathfrak{C}$ is the language UTVPI, and it is $\mathsf{co\text{-}NP}$-complete when $\mathfrak{C}$ is the language bUTVPI. For these languages, our characterisation yields an exponential-time algorithm for deciding the losslessness of Σ under Δ by a number of unsatisfiability checks in $\mathfrak{C}$ which is exponentially bound by the size of Δ.

By definition, Σ is lossless under Δ if the extension of R under every model I of $\Sigma \cup \Delta$ coincides with the union of the extensions of each V_i under I, that is, $R^I = {V_1}^I \cup \dots \cup {V_n}^I$. As the extension of each view symbol is always included in the extension of R, the problem is equivalent to checking that there is no model I of $\Sigma \cup \Delta$ where a tuple $\overline{t} \in R^I$ does not belong to any ${V_i}^I$, which in turn means that for each selection in Σ, some value in $\overline{t}$ at a non-interpreted position does not satisfy an equality or inequality, or the values in $\overline{t}$ at interpreted positions do not satisfy the $\mathfrak{C}$-formula σ. As already noted for CDCs, also the formulae in Σ apply to one tuple at a time, so we can again focus on single tuples. With each equality and inequality we associate as before a propositional variable, whose evaluation determine whether the equality or inequality is satisfied. We then need to check that there exists no valuation α that does not satisfy some of the equalities and inequalities in each selection, but satisfies the $\mathfrak{C}$-formulae in the r.h.s. of all the $\mathfrak{C}$-CDCs that are applicable under α, along with the *negation* of any $\mathfrak{C}$-constraint appearing in some selection of Σ whose equalities and inequalities are satisfied by α. Indeed, from such a valuation and corresponding assignment of values from **idom** satisfying the relevant $\mathfrak{C}$-formulae, we can construct a tuple that provides a counterexample to losslessness.

Before discussing the details, let us first extend the setting of Example 1, by introducing a horizontal decomposition which will serve as our running example throughout this section.

Example 4. Consider the same R as in Example 1, let $\mathbf{V} = \{ V_1, V_2, V_3 \}$, and let Σ be the horizontal decomposition defined as follows:

$$V_1 \colon x_2 \neq a \wedge x_3 = b \;; \qquad V_2 \colon y_2 < 4 \;; \qquad V_3 \colon x_3 \neq b \;.$$

Intuitively, V_1, V_2 and V_3 select employees working as managers in departments other than ICT, getting strictly less than 4 as bonus, and not working as managers, respectively.

Let Π_Δ be the propositional theory associated with Δ as in Sec. 3. Similarly, we construct a propositional theory Π_Σ associated with Σ as follows: For each $\varphi \in \Sigma$, which we recall has the form (2), we build a propositional formula of the form (3), where v is either a fresh propositional variable (not in Π_Δ) associated with the $\mathfrak{C}$-formula $\sigma(\overline{y})$, if any, occurring in φ, or $\bot$ otherwise. This is because the selections in Σ may consist only of equalities and inequalities without a $\mathfrak{C}$-constraint, whereas $\mathfrak{C}$-CDCs must have a $\mathfrak{C}$-formula in the consequent in order to be meaningful.

Let $\Pi = \Pi_\Delta \cup \Pi_\Sigma$ and observe that each propositional formula in Π is of the form $\top \wedge L \to v \vee \bot$, in which L is a (possibly empty) conjunction of literals and v (if present) is a propositional variable associated with a $\mathfrak{C}$-constraint, denoted by $\mathsf{constr}(v)$ as before. We omit $\top$ in the l.h.s. of the implication if L is non-empty, and $\bot$ in the r.h.s. when v is present. We consider an *extended* auxiliary theory $\Pi_\bot$ for Π defined as follows:

$$\Pi_\bot = \left\{ p_i^a \wedge p_i^b \to \bot \mid a \neq b,\ p_i^a, p_i^b \in \mathsf{pvar}(\Pi) \right\} \cup \left\{ P \in \Pi_\Sigma \mid P = L \to \bot \right\}, \quad (6)$$

where the first set in the union is the same as in (4), but on $\mathsf{pvar}(\Pi)$ rather than just $\mathsf{pvar}(\Pi_\Delta)$, and the second is the set of all the propositional formulae in Π_Σ whose r.h.s. is not associated with a $\mathfrak{C}$-constraint.[1]

Example 5. The propositional theory associated with the horizontal decomposition Σ of Example 4 is $\Pi_\Sigma = \{ \neg p_2^a \wedge p_3^b \to \bot,\ \top \to v_4,\ \neg p_3^b \to \bot \}$. Let Π_Δ be the propositional theory of Example 2, and let $\Pi = \Pi_\Delta \cup \Pi_\Sigma$. Then, the association between the propositional variables in $\mathsf{cvar}(\Pi)$ and UTVPIs is as in Example 2 but extended with $\{ v_4 \mapsto y_2 < 4 \}$, and the extended auxiliary theory for Π is $\Pi_\bot = \{ \neg p_2^a \wedge p_3^b \to \bot,\ \neg p_3^b \to \bot \}$.

For a valuation α of $\mathsf{pvar}(\Pi_\Delta)$, let Π_Δ^α be the α-filtering of Π_Δ defined as in Sec. 3. Similarly, for a valuation α of $\mathsf{pvar}(\Pi_\Sigma)$, the α-filtering of Π_Σ is

$$\Pi_\Sigma^\alpha = \{ \neg\, \mathsf{constr}(v) \mid (\top \wedge L \to v) \in \Pi_\Sigma,\ L \text{ is empty or } \alpha(L) = \mathtt{T} \}\ , \quad (7)$$

consisting of the *negation* of $\mathfrak{C}$-constraints associated with propositional variables that occur in some formula of Π_Σ, whose l.h.s. holds true under α. Observe that in (7), differently from (5), $\mathfrak{C}$-constraints are negated because a counter-instance I to losslessness is such that R^I consists of only one tuple and ${V_1}^I \cup \cdots \cup {V_n}^I = \varnothing$, and therefore, whenever all of the equalities and inequalities in the selection that defines a view are satisfied by I, the $\mathfrak{C}$-constraint is not. On the other hand, the $\mathfrak{C}$-constraint in the consequent of a CDC must hold whenever the equalities and inequalities in the antecedent are satisfied. For a valuation α of $\mathsf{pvar}(\Pi)$, the α-filtering of Π is the set $\Pi^\alpha = \Pi_\Delta^\alpha \cup \Pi_\Sigma^\alpha$. Note that, as $\mathfrak{C}$ is closed under negation, all the constraints in Π^α are $\mathfrak{C}$-formulae.

[1] These originate from formulae of Σ that do not specify a $\mathfrak{C}$-constraint $\sigma(\overline{y})$.

The main result of this section gives a characterisation of lossless horizontal decompositions in terms of unsatisfiability in $\mathfrak{C}$.

Theorem 3. *Let Δ consist of $\mathfrak{C}$-CDCs and let Σ be a horizontal decomposition. Let Π_Δ and Π_Σ be the propositional theories associated with Δ and Σ, respectively, and let $\Pi_\perp$ be the extended auxiliary theory for $\Pi = \Pi_\Delta \cup \Pi_\Sigma$. Then, Σ is lossless under Δ if and only if the α-filtering $\Pi^\alpha = \Pi^\alpha_\Delta \cup \Pi^\alpha_\Sigma$ of Π is unsatisfiable for every valuation α of* $\mathsf{pvar}(\Pi)$ *satisfying $\Pi_\perp$.*

Obviously, whenever the satisfiability of sets of $\mathfrak{C}$-formulae is decidable, Theorem 3 directly yields an algorithm for deciding whether a horizontal decomposition is lossless. We illustrate this in our running example with UTVPIs.

Example 6. Consider Π and $\Pi_\perp$ from Example 5. The only valuation of $\mathsf{pvar}(\Pi)$ satisfying $\Pi_\perp$ is $\alpha = \{\, p_2^a \mapsto \mathtt{T},\, p_3^b \mapsto \mathtt{T}\}$, for which $\Pi^\alpha_\Delta = \{\, y_1 + y_2 \leq 5,\, y_2 \geq 2,\, y_1 - y_2 \geq 0 \,\}$ and $\Pi^\alpha_\Sigma = \{\, y_2 \geq 4 \,\}$. Note that $y_2 \geq 4$ in Π^α_Σ is $\neg\,\mathsf{constr}(v_4)$, that is, the negation of $y_2 < 4$. The set $\Pi^\alpha = \Pi^\alpha_\Delta \cup \Pi^\alpha_\Sigma$ is unsatisfiable because from $y_1 + y_2 \leq 5$ and $y_2 \geq 4$ we obtain $y_1 \leq 1$, which together with $y_1 - y_2 \geq 0$ yields $y_2 \leq 1$, in conflict with $y_2 \geq 2$.

The *losslessness problem* in $\mathfrak{C}$ is the problem that takes as input a horizontal decomposition Σ specified by $\mathfrak{C}$-selections and a set of $\mathfrak{C}$-CDCs Δ, and answers the question: "Is Σ lossless under Δ?" In light of Theorem 3, the unsatisfiability problem for $\mathfrak{C}$, asking whether a given set F of $\mathfrak{C}$-formulae is unsatisfiable, reduces to the losslessness problem in $\mathfrak{C}$ as follows: take Δ empty and, for each $\mathfrak{C}$-formula in F, consider a new view symbol defined in Σ by an $\mathfrak{C}$-selection consisting only of the negation of that formula (which is still in $\mathfrak{C}$ as $\mathfrak{C}$ is closed under negation) and without equalities and inequalities. Thus, whenever the unsatisfiability problem in $\mathfrak{C}$ is C-hard, for some complexity class C, then the losslessness problem in $\mathfrak{C}$ is also C-hard. We have the following complexity results concerning the losslessness problem in UTVPI and bUTVPI.

Theorem 4. *The losslessness problem in $\mathfrak{C}$ is in* co-NP *when $\mathfrak{C} =$* UTVPI*, and is* co-NP*-complete when $\mathfrak{C} =$* bUTVPI*.*

5 Adding Functional and Inclusion Dependencies

In this section, we will study the interaction between the newly-introduced CDCs and traditional database constraints w.r.t. the losslessness of horizontal decompositions. This investigation is crucial in order to understand to what extent the techniques we described in Sec. 4 can be applied to an existing database schema, on which a set of integrity constraints besides CDCs is already defined. We will focus two well-known classes of integrity constraints, namely functional dependencies (FDs) and unary inclusion dependencies (UINDs), and show how to fully capture the interaction between them and CDCs w.r.t. lossless horizontal decomposition, under certain syntactic restrictions. It is important to remark that we consider restrictions solely on the CDCs, so that existing integrity constraints need not be changed in order to allow for CDCs.

We begin by observing that FDs do not interact with CDCs, and can thus be freely allowed in combination with them, as long as lossless horizontal decomposition is concerned.

Theorem 5. *Let Δ consist of FDs and CDCs, and let $\Delta' \subseteq \Delta$ consist of all the CDCs in Δ. Then, a horizontal decomposition is lossless under Δ if and only if it is lossless under Δ'.*

The above follows from the fact that a horizontal decomposition is lossy under CDCs precisely if there is a one-tuple counter-example to its losslessness, while an FD violation always involves at least two tuples.

Let us recall that an instance I satisfies a UIND $R[i] \subseteq R[j]$ if every value in the i-th column of R^I appears in the j-th column of R^I. While FDs do not play any role in whether a horizontal decomposition is lossless, this is definitely not the case for UINDs, as the following example shows.

Example 7. Let $\mathbf{R} = \{R\}$ and $\mathbf{V} = \{V\}$, with $|R| = |V| = 2$ and both positions interpreted over the integers. Consider the horizontal decomposition Σ of $\mathbf{R}$ into $\mathbf{V}$ defined by $V\colon y_1 > 3$, and integrity constraints Δ on $\mathbf{R}$ given by $\top \rightarrow y_2 > 3$ along with the UIND $R[1] \subseteq R[2]$. It is easy to see that Δ entails the additional CDC $\top \rightarrow y_1 > 3$. Therefore, Σ is lossless as V selects all the tuples in R, which is clearly not the case in the absence of the UIND.

Notation. As it might not be possible to compare values from **dom** with values from **idom** and vice versa, in our setting we consider UINDs on positions that are either both interpreted or both non-interpreted. We introduce the following notation for UINDs: we write $R[x_i] \subseteq R[x_j]$ with $i, j \in \{1, \dots, k\}$, where k is the number of non-interpreted positions of R, to denote the UIND $R[i] \subseteq R[j]$; we write $R[y_i] \subseteq R[y_j]$ with $i, j \in \{|R| - k, \dots, |R|\}$ to denote $R[i + k] \subseteq R[j + k]$.

As we have seen in Example 7 in the case of UINDs, when we allow CDCs in combination with constraints from another class, their interaction may entail additional constraints which may in turn influence losslessness. We now introduce a general property, called *separability*, for which the interaction of the CDCs with other constraints can be fully captured, as long as lossless horizontal decomposition is concerned, so that after making explicit the result of such interaction we can "separate" the constraints that are not CDCs.

Definition 2 (Separability). *Let C be a class of integrity constraints, let $\mathcal{S}$ be a finite set of sound inference rules*[2] *for C extended with CDCs, and let Δ consist of CDCs and C-constraints. We say that the C-constraints are $\mathcal{S}$-*separable *in Δ from the CDCs if every horizontal decomposition is lossless under Δ precisely when it is lossless under Δ', where Δ' is obtained by first saturating Δ with $\mathcal{S}$*

[2] We assume the reader to be familiar with the standard notions (from proof theory) of inference rule and soundness.

and then retaining only the CDCs.[3] *When the* C*-constraints are* $\mathcal{S}$*-separable for* $\mathcal{S} = \varnothing$*, we simply say that they are* separable*.*

According to the above definition, we have that the FDs are separable from the CDCs in every set consisting of FDs and CDCs, in light of Theorem 5. Observe also that $\mathcal{S}$-separability implies $\mathcal{S}'$-separability for every sound $\mathcal{S}' \supseteq \mathcal{S}$.

The interaction of UINDs on interpreted attributes with a restricted form of CDCs is captured by the following *domain propagation* rule:

$$\frac{\top \rightarrow \delta(y_i) \qquad\qquad R[y_j] \subseteq R[y_i]}{\top \rightarrow \delta(y_j)}\ , \tag{dp}$$

whose soundness can be easily shown.

It turns out that if all the CDCs that mention variables corresponding to the interpreted positions in some UIND have the form used in (dp), then the domain propagation rule fully captures the interaction between such CDCs and UINDs on interpreted positions w.r.t. the losslessness of horizontal decompositions.

Theorem 6. *Let Δ be a set of UINDs on interpreted positions and CDCs such that, for every UIND $R[y_i] \subseteq R[y_j]$ in Δ, all of the CDCs in Δ that mention the variable y, where y is y_i or y_j, are of the form $\top \rightarrow \delta(y)$. Then, the UINDs are* $\{$(dp)$\}$*-separable in Δ from the CDCs.*

We now turn our attention to UINDs on non-interpreted positions in combination with CDCs. We introduce syntactic restrictions, one on the CDCs w.r.t. the UINDs and one on the CDCs in themselves, that together ensure the separability of the UINDs.

Definition 3. *Let Δ consist of CDCs and UINDs. We say that the CDCs are* non-overlapping *with the UINDs on non-interpreted positions if for each UIND $R[x_i] \subseteq R[x_j]$ and any two CDCs $\phi(\overline{x}_1, \overline{y}_1)$ and $\psi(\overline{x}_2, \overline{y}_2)$ in Δ such that either*

a) $x_i \in \mathsf{var}(\overline{x}_1)$ *and* $x_j \in \mathsf{var}(\overline{x}_2)$*, or*
b) $\mathsf{var}(\overline{x}_1) = \varnothing$*, and* $x_i \in \mathsf{var}(\overline{x}_2)$ *or* $x_j \in \mathsf{var}(\overline{x}_2)$*,*

it is the case that $\mathsf{var}(\overline{y}_1) \cap \mathsf{var}(\overline{y}_2) = \varnothing$*.*

Intuitively, the above requires that CDCs ϕ and ψ, such that the antecedent of ϕ mentions the variable x_i affected by a UIND $R[x_i] \subseteq R[x_j]$ and the antecedent of ψ mentions x_j, have no overlap in the variables mentioned in their consequents. This must also hold when one of the CDCs has the form $\top \rightarrow \delta(\overline{y})$ and the other mentions one of the variables affected by the UIND. The UTVPI-CDC and the UIND in Example 7 satisfy the non-overlapping restriction.

Definition 4. *Let Δ consist of CDCs. We say that Δ is* partition-free *if it is not possible to find n distinct variables $\overline{x}[k_1], \dots, \overline{x}[k_n] \in \mathsf{var}(\overline{x})$, constants $a_1, \dots, a_n$ from* **dom** *and 2^n distinct CDCs $\varphi_1, \dots, \varphi_{2^n}$ in Δ such that:*

[3] As the constraints that are not CDCs are in any case filtered out after saturating Δ with $\mathcal{S}$, it does not matter whether C extended with CDCs is closed under $\mathcal{S}$ or not.

a) the antecedent of each CDC mentions only $\overline{x}[k_1], \ldots, \overline{x}[k_n]$ *without repetitions and, for every* $i \in \{1, \ldots, n\}$*, it contains either the equality* $\overline{x}[k_i] = a_i$ *or the inequality* $\overline{x}[k_i] \neq a_i$*;*

b) $\{0, \ldots, 2^{n-1}\} \neq \{d \in \mathbb{N} \mid \mathrm{enc}(\varphi_j)$ *is the binary encoding of* d, $1 \leq j \leq 2^n\}$, *where* $\mathrm{enc}(\varphi_j)$ *is an* n*-digit binary number* b *whose* i*-th digit is* 1 *if* $\overline{x}[k_i] = a_i$ *is in* φ_j *and* 0 *otherwise.*

Intuitively, the above requires that there cannot be 2^n CDCs, whose antecedents are all on the same n variables and constants and cover all possible combinations of equalities and inequalities between a variable and the corresponding constant. We remark that for every partition-free set of CDCs it is always possible to find a valuation under which the antecedents of all of its CDCs evaluate to false.

Theorem 7. *Let* Δ *consist of UINDs on non-interpreted positions and CDCs, where the CDCs are partition-free and non-overlapping with the UINDs. Then, the UINDs are separable in* Δ *from the CDCs.*

We close this section by remarking that the restrictions of Theorems 6 and 7 can be put together in order to combine CDCs with UINDs on both interpreted and non-interpreted positions.

6 Discussion and Outlook

In this paper, we investigated lossless horizontal decomposition under integrity constraints in a setting where some of the attributes in the schema are interpreted over specific domains. Such domains are associated with special predicates and functions that allow to compare data values other ways beyond equality. To the best of our knowledge, this scenario had not yet been explored in the literature, in particular in the study of horizontal decomposition, which has mostly been concerned with uninterpreted data. In this context, we introduced a new class of integrity constraints, called CDCs, and we have shown how to check for the losslessness of horizontal decompositions under CDCs in isolation, as well as in combination with FDs and UINDs.

Even though the focus of this paper has been on domain constraints on interpreted positions, observe that domain constraints on non-interpreted positions of the form $\top \rightarrow x_i = a_1 \vee \cdots \vee x_i = a_n$, with $a_1, \ldots, a_n \in \mathbf{dom}$, can be expressed by means of two $\mathfrak{C}$-CDCs:[4]

$$x_i \neq a_1 \wedge \cdots \wedge x_i \neq a_n \rightarrow \delta(\overline{y}) \ ; \qquad x_i \neq a_1 \wedge \cdots \wedge x_i \neq a_n \rightarrow \neg\delta(\overline{y}) \ ;$$

for some $\delta(\overline{y}) \in \mathfrak{C}$.[5] Being standard CDCs, such constraints are directly handled by the techniques we described for the consistency of CDCs and the losslessness of horizontal decompositions.

[4] Repetition of the same variable in the antecedent of a CDC is allowed.

[5] Recall that $\mathfrak{C}$ is closed under negation, hence $\neg\delta(\overline{y}) \in \mathfrak{C}$.

The results presented in Sec. 5 about separable combinations of CDCs and UINDs do not automatically carry over to the case in which FDs are also present. Indeed, even though FDs do not interact directly with CDCs, they do in general interact with UINDs, which in turn interact with CDCs. The only case in which there is no interaction between FDs and UINDs is when the former are on non-interpreted positions while the latter are on interpreted ones. In fact, Theorem 6 can be straightforwardly generalised to include also FDs on non-interpreted positions. In the other cases, the interaction between FDs and UINDs can be fully captured, as there is a sound and complete axiomatization for finite implication of FDs and UINDs [1]. Therefore, we conjecture that FDs and UINDs together are $\mathcal{S}$-separable from (appropriately restricted) CDCs, where $\mathcal{S}$ consists of the known inference rules for FDs and UINDs extended with our domain propagation rule, and that Theorems 6 and 7 can be generalised to include also FDs (on all attributes).

Applying a general criterion for the translatability of view updates, recently provided in [6], it is possible to determine whether an update can be performed on each fragment, and propagated to the underlying database, without affecting the other fragments. The same information can also be used for adding suitable conditions in the selections defining the view symbols so that each fragment is disjoint with the others, that is, to obtain a partition. In follow-up work, we plan on performing an in-depth study of partitioning and update propagation in the setting studied in this paper.

A lossy horizontal decomposition can always be turned into a lossless one by defining an additional fragment that selects the missing tuples. We are currently working on a general algorithm to compute the definition, in the setting studied in this paper, of the unique fragment that selects all and only the rows of the original relation which are not selected by any of the other fragments.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Bra, P.D.: Horizontal Decompositions in the Relational Database Model. Ph.D. thesis, University of Antwerp (1987)
3. Ceri, S., Negri, M., Pelagatti, G.: Horizontal data partitioning in database design. In: SIGMOD Conference, pp. 128–136. ACM Press (1982)
4. De Bra, P.: Horizontal decompositions based on functional-dependency-set-implications. In: Atzeni, P., Ausiello, G. (eds.) ICDT 1986. LNCS, vol. 243, pp. 157–170. Springer, Heidelberg (1986)
5. De Bra, P., Paredaens, J.: Conditional dependencies for horizontal decompositions. In: Díaz, J. (ed.) ICALP 1983. LNCS, vol. 154, pp. 67–82. Springer, Heidelberg (1983)
6. Franconi, E., Guagliardo, P.: On the translatability of view updates. In: AMW 2012. CEUR-WS, vol. 866, pp. 154–167 (2012)
7. Furtado, A.L.: Horizontal decomposition to improve a non-BCNF scheme. SIGMOD Record 12(1), 26–32 (1981)

8. Jaffar, J., Maher, M.J., Stuckey, P.J., Yap, R.H.C.: Beyond finite domains. In: Borning, A. (ed.) PPCP 1994. LNCS, vol. 874, pp. 86–94. Springer, Heidelberg (1994)
9. Lahiri, S.K., Musuvathi, M.: An efficient decision procedure for UTVPI constraints. In: Gramlich, B. (ed.) FroCos 2005. LNCS (LNAI), vol. 3717, pp. 168–183. Springer, Heidelberg (2005)
10. Maier, D., Ullman, J.D.: Fragments of relations. In: SIGMOD Conference, pp. 15–22. ACM Press (1983)
11. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, 3rd edn. Springer (2011)
12. Schutt, A., Stuckey, P.J.: Incremental satisfiability and implication for UTVPI constraints. INFORMS Journal on Computing 22(4), 514–527 (2010)
13. Seshia, S.A., Subramani, K., Bryant, R.E.: On solving boolean combinations of UTVPI constraints. JSAT 3(1-2), 67–90 (2007)

MatchBench: Benchmarking Schema Matching Algorithms for Schematic Correspondences

Chenjuan Guo, Cornelia Hedeler, Norman W. Paton, and Alvaro A.A. Fernandes

School of Computer Science, University of Manchester, M13 9PL, UK
{guoc,chedeler,norm,alvaro}@cs.man.ac.uk

Abstract. Schema matching algorithms aim to identify relationships between database schemas, which are useful in many data integration tasks. However, the results of most matching algorithms are expressed as semantically inexpressive, 1-to-1 associations between pairs of attributes or entities, rather than semantically-rich characterisations of relationships. This paper presents a benchmark for evaluating schema matching algorithms in terms of their semantic expressiveness. The definition of such semantics is based on the classification of schematic heterogeneities of Kim *et al.*. The benchmark explores the extent to which matching algorithms are effective at diagnosing schematic heterogeneities. The paper contributes: (i) a wide range of scenarios that are designed to systematically cover several reconcilable types of schematic heterogeneities; (ii) a collection of experiments over the scenarios that can be used to investigate the effectiveness of different matching algorithms; and (iii) an application of the experiments for the evaluation of matchers from three well-known and publicly available schema matching systems, namely COMA++, Similarity Flooding and Harmony.

1 Introduction

Schema matching methods identify *matches* between elements of data sources that show similar properties (e.g., names, instances and structures) [4, 20]. Matching methods are not an end in themselves, but rather form part of other operations, such as *schema mapping* that refines matches into declarative but executable mappings (e.g., in SQL or XSLT) to specify the relationships between the data sources [11, 5]. Schema matching and mapping are important because a wide range of information management and integration tasks [13, 12], such as data exchange, evolution and distributed query processing, depend on a detailed understanding of the relationships between data sources.

Such integration tasks must be built on appropriate executable mappings, which, in turn, require clear characterisations of matches between data sources. However, although matches may be generated by a large number of different techniques, they are often numerous, uncertain and conflicting [3]. As such, when evaluating matches, it seems important to explore what kind of information is carried by matches that must be taken into account by executable programs.

We note that there have been several evaluation activities relating to schema matching/mapping in the data integration community in recent years, such as Ontology Alignment Evaluation Initiative (OAEI) [1], XBenchmark [10], eTuner [15] and STBenchmark [2]. The first three activities aim to evaluate schema or ontology matching

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 92–106, 2013.

systems (e.g., [8, 17, 7]) in terms of accuracy or correctness, e.g., *precision*, *recall* and *F-measure*, of matches identified by the existing matching systems, while STBenchmark aims to compare schema mapping systems (e.g., [11, 5, 6]) in terms of the effectiveness of the support provided to mapping developers.

In this paper, we present a benchmark, called MatchBench, with the aim of understanding the effectiveness of schema matching systems in identifying *specific relationships* between elements of two schemas, rather than simply assessing the correctness of matches identified by the matching systems, thus differentiating MatchBench from previous evaluation activities. We characterise such relationships between schema elements using the classification of schematic heterogeneities of Kim *et al.* [14]. We do not intend to enumerate all kinds of relationships in MatchBench but try to exemplify a collection of semantic relationships based on the schematic heterogeneities.

Thus, the hypothesis behind MatchBench is that the effectiveness of matching systems in practice can be evaluated in terms of their ability to diagnose (or support the diagnosis of) such schematic heterogeneities as those proposed by Kim *et al.* The contributions of the paper are presented as follows:

1. A collection of scenarios, based on the schematic heterogeneities of Kim *et al.* [14], that systematically vary the amount and nature of the evidence available about heterogeneities.
2. An experiment design over the scenarios at (1) that can be used as a benchmark to investigate the contexts within which specific matchers are more, or less, effective.
3. The application of the benchmark to schema matching techniques is supported within three well-known and publicly available matching systems[1], namely COMA++ [8], Similarity Flooding [17] and Harmony [22].

The remainder of the paper is structured as follows. Section 2 introduces the schematic heterogeneities of Kim *et al.* [14]. Section 3 describes MatchBench, including the offered scenarios and the associated experiments. Sections 4 describes the application of MatchBench to matchers provided by COMA++, Similarity Flooding and Harmony, respectively. Section 5 reviews related work, in particular on the evaluation of schema matching techniques. Section 6 presents some overall conclusions.

2 Schematic Correspondences

The schematic heterogeneities proposed by Won Kim *et al.* [14] are defined as different symbolic representations of data that represent the same real world information. We essentially use the terms heterogeneity and correspondence as synonyms – a heterogeneity is an inconsistency between data sources in representation, and a correspondence is a description of the heterogeneity that allows it to be managed.

In this paper, we adopt the classification of schematic correspondences between relational schemas proposed by Won Kim *et al.* [14], and have refined the characteristics of many-to-many entity correspondences from [14] to distinguish horizontal and vertical partitioning. Before moving on to the details, let the following be the schemas of two

[1] In order to maintain the initial feature of matching systems, we decided not to re-implement the matching systems that are not publicly available.

independently designed relational databases RDB1 and RDB2[2].

RDB1:
home_cust (id*, name, birth, a_id$^+$, p_city, p_area, p_local)
oversea_cust (id*, name, birth, a_id$^+$, p_city, p_area, p_local)
account (id*, name, balance, tax)

RDB2:
customer (id*, c_fname, c_lname, c_birth, account_id$^+$)
cust_phone (id^{*+}, city, area, local, extension)
cust_account (id*, account_name, account_balance)

Both RDB1 and RDB2 contain information about customers and their accounts. Even though they represent the information differently, it can be identified that they represent broadly the same real world information, and that correspondences exist between them at both entity and attribute levels:
(i) The *entity-level correspondences* indicate the equivalence between two (sets of) entities (e.g., tables), which can be decomposed into one-to-one and many-to-many entity correspondences, where

- *one-to-one entity correspondences* relate pairwise entities that represent the same information. For example, account in RDB1 and cust_account in RDB2 can be considered equivalent but show the following heterogeneities:
  - *name conflict*, which indicates that equivalent entities have different names. In the following, this conflict is called Different Names for the Same Entities (*DNSE*). When different entities happen to have the same name, we call the conflict Same Name for Different Entities (*SNDE*).
  - *missing attributes conflict*, which identifies attributes that are present in one entity but not in the other (e.g., attribute tax in account is a missing attribute of cust_account).
- *many-to-many entity correspondences* relate two sets of entities that represent the same information. For example, home_cust and oversea_cust together in RDB1 describe the same information about customers as customer and cust_phone in RDB2. It can be seen that these two sets of entities in RDB1 and RDB2 do not have the same structure, but the underlying information is similar. This difference results in distinct types of many-to-many conflicts. Borrowing terminology from distributed database systems [19],we classify them as follows:
  - *horizontal partitioning* (*HP*), where one single entity is partitioned along its instances into multiple entities in another schema. As such, all attributes of the former entity are present in each of the corresponding entities in the latter (e.g., home_cust and oversea_cust in RBD1).
  - *vertical partitioning* (*VP*), where a single entity is partitioned into multiple entities in another schema, where the attributes in each of the latter constitute subsets of the attributes in the former. The primary key of the vertically partitioned entity appears as an equivalent attribute in every one of its vertical partitions in the other schema, whereas other attributes of the former entity are present only once in the latter (e.g., customer and cust_phone in RDB2).

[2] Symbols * and $^+$ indicate primary key and foreign key attributes, respectively.

Given the above information, we are then able to enumerate 4 types of many-to-many entity correspondences: *HP* vs *HP*, *HP* vs *VP*, *VP* vs *HP* and *VP* vs *VP* correspondences. For example, the correspondence between entity sets {home_cust, oversea_cust} and {customer, cust_phone} is a *HP* vs *VP* correspondence. Readers may follow the definitions to enumerate other partitioning types, such as hybrid partitioning, which refers to the cases where HP and VP appear together in the source (or target).

(ii) The *attribute-level correspondences* indicate the equivalence between two (sets of) attributes. For the remainder of the paper, we assume that attributes associated by attribute-level correspondences belong to entities that are participating in some entity-level correspondence. Similar the entity-level correspondence, the attribute-level correspondences can be decomposed into one-to-one and many-to-many correspondences, where

- *one-to-one attribute correspondences* relate pairwise attributes. Equivalent attributes may have different names, so such a conflict is called Different Names for the Same Attributes (*DNSA*) (e.g., account.name in RDB1 and cust_account.account_name in RDB2). By contrast, attributes that are different may have the same name, giving rise to Same Name for Different Attributes (*SNDA*) correspondences.
- *many-to-many attribute correspondences* associate two sets of attributes that present the same property of equivalent entities. For example, the single attribute home_cust.
name in RDB1 and the set of attributes customer.c_fname and customer.c_lname in RDB2 represent names of customers.

3 Benchmark Description

This section describes the benchmark[3] consisting of: (i) a collection of scenarios, in which schematic heterogeneities are systematically injected into an initial database; and (ii) a collection of experiments that investigate the ability of matching methods to identify specific properties exhibited by the scenarios.

3.1 Scenario

The scenarios build upon the *source* and *target* databases illustrated in Fig. 1, derived from TPC-E (http://www.tpc.org/tpce/), which are manipulated in a controlled manner.

There are *positive* and *negative* scenarios. In the *positive* scenarios, the starting point is that the *source* and *target* databases have a single table in common, into which schematic heterogeneities described in Section 2 are systematically introduced. In the implementation, the initial *target* database is extended with a single table from the *source* (see Fig. 1). In the *negative* scenarios, the starting point is that the source and

[3] This paper only means to demonstrate a general idea of MatchBench. Readers may find the complete version for all available scenarios and experiments from
`http://code.google.com/p/matchbench/`.

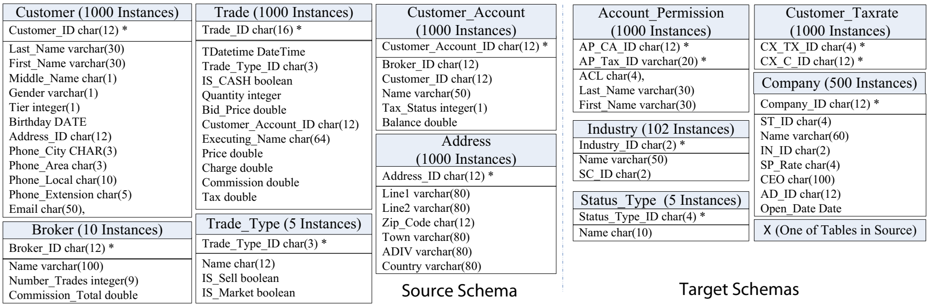


Fig. 1. The *source* and *target* databases used as a basis for scenario generation, where primary keys are marked with $*$

target databases have no single table in common, but similarities have been systematically introduced, giving rise to scenarios where tables should not be matched, but where there are some similarities between tables.

Positive Scenarios for One-to-One Entity Correspondences. Fig. 2 describes the space of positive scenarios where heterogeneities are introduced into one-to-one identical entities[4]. In the figure, boxes represent scenario sets and arrows represent the systematic introduction of heterogeneities into the scenario sets. Each *scenario set* consists of a collection of databases each of which manifests examples of the heterogeneities named in the corresponding box, the definitions of which are provided below. For example, *Scenario Set 1* represents the starting point for the introduction of the heterogeneities, and the arrow leading to *Scenario Set 5* indicates that it has been derived from *Scenario Set 1* through the changing of entity names.

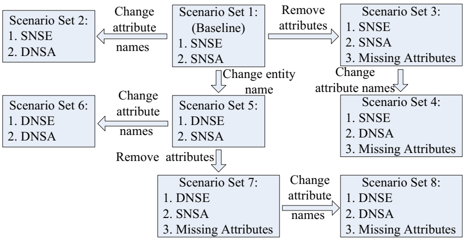


Fig. 2. Positive scenarios for one-to-one entity correspondences

In what follows, where names are described as the *same* they are identical, and where they are described as *similar* their strings overlap; neither of these properties hold for *different* names. Following the terminology introduced in Section 2, terms used in Fig. 2 include *SNSE* as Same Name for Same Entity; *DNSE* as Different Names for Same Entity; *SNSA* as Same Name for Same Attribute; and *DNSA* as Different Names for Same Attribute. As such, a scenario set that exhibits one-to-one entity heterogeneities may also exhibit one-to-one attribute heterogeneities.

In each scenario set, the extents of equivalent entities either contain the *same instances* (SI) or *disjoint instances* (DI). The *disjoint instances* are generated by partitioning instances of an original entity into two disjoint sets of instances, thus forming

[4] The order of introducing different types of heterogeneities is insignificant.

disjoint instances of two equivalent entities. Overlapping instances are also possible real world cases, but are not implemented in MatchBench.

Negative Scenarios for One-to-One Entity Correspondences. The space of negative scenarios for one-to-one different entities, where pairs of entities represent different real world information, is described in Fig. 3. Terms used to describe the properties of the scenario sets include *DNDE* as Different Names for Different Entities; *SNDE* as Same Name for Different Entities; *DNDA* as Different Names for Different Attributes; *SNSA* as Same Name for Same Attribute; *DNSA* as Different Names for the Same Attributes; and *SNDA* as Same Name for Different Attributes.

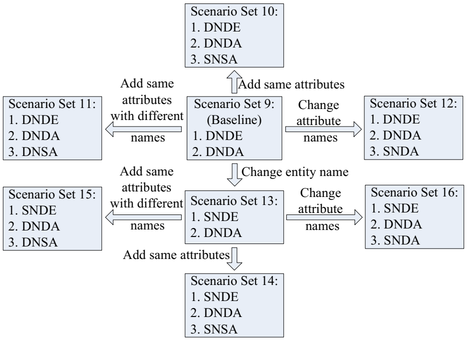


Fig. 3. Negative scenarios for one-to-one entity correspondences

Positive Scenarios for Attribute Many-to-One Correspondences. In Fig. 4, the space of attribute many-to-one correspondences is described, where a set of attributes and a single attribute that belong to equivalent entities represent the same real world information. We note that most schema matchers only handle many-to-one attribute correspondences, and thus we set up a task that existing matchers can manage. MatchBench includes three different types of attribute many-to-one correspondences shown as follows.

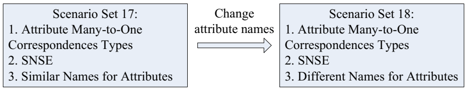

Scenario Set 17:
1. Attribute Many-to-One Correspondences Types
2. SNSE
3. Similar Names for Attributes
Change attribute names
Scenario Set 18:
1. Attribute Many-to-One Correspondences Types
2. SNSE
3. Different Names for Attributes

Fig. 4. Positive scenarios for attribute many-to-one correspondences.

1. numeric operation: $(\mathit{price} + \mathit{charge} + \mathit{commission}) \times (1 + \mathit{tax}) = \mathit{price}$
2. string concatenation: Concat $(\mathit{first\_name}, \mathit{middle\_name}, \mathit{last\_name}) = \mathit{name}$
3. numeric concatenation:
   Concat $(\mathit{phone\_city}, \mathit{phone\_area}, \mathit{phone\_local}, \mathit{phone\_extension}) = \mathit{phone}$

Similar to Fig. 2, extents of equivalent entities are generated that give rise to SI and DI cases for scenario set 17. Scenario set 18 only contains SI cases but not DI, in order to retain a certain level of similarity between attributes.

Positive Scenarios for Entity Many-to-Many Correspondences. Two sets of entities, shown in Fig. 5, represent the same real world information. Three different types of many-to-many entity correspondences are included in MatchBench:

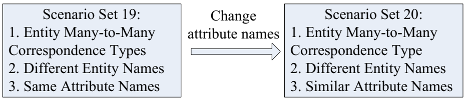

Scenario Set 19:
1. Entity Many-to-Many Correspondence Types
2. Different Entity Names
3. Same Attribute Names
Change attribute names
Scenario Set 20:
1. Entity Many-to-Many Correspondence Type
2. Different Entity Names
3. Similar Attribute Names

Fig. 5. Positive scenarios for many-many entity correspondences

- *HP vs HP*, where the two sets are related by horizontal partitioning.
- *VP vs VP*, where the two sets are related by vertical partitioning.
- *HP vs VP*, where the two sets are related by horizontal and vertical partitioning.

3.2 Experiments

Effectiveness Measures. The effectiveness of matching systems is evaluated in MatchBench by identifying whether the systems meet specific requirements for diagnosing each type of schematic heterogeneity. Each such requirement is met if results of the systems are close to the correct matches provided by the scenarios presented in this section. Following terms in the standard definitions [9] of information retrieval, we call the correct matches as ground truth. We compare the results of the systems with the ground truth, and report recall, precision and F-measure of the results following to shown the effectiveness of the matching systems.

Experiment Design. Building on the scenarios, we designed 10 experiments to measure how effectively matching systems identify the presence of schematic heterogeneities. Due to the space limitations, only 4 experiments are presented in this paper. More experiments are included in the technical report.

In Experiments 1, 3 and 4, where schematic heterogeneities are exhibited in the chosen scenarios, the F-measure is reported in the vertical axis drawn in the figures produced in Section 4.2. The higher the F-measure reports, the better is the matching system for diagnosing the heterogeneity. In Experiment 2, which involves negative scenarios, where there are no such heterogeneities in the chosen scenarios, 1 – F-measure is reported on the vertical axis so that larger values also reflect the better effectiveness of the matching system on not reporting the heterogeneities.

Experiment 1: *Identifying when the same entity occurs in positive scenarios.* This experiment involves *Scenario Sets 1* to *8* in Fig. 2, and reports on the ability of the matchers to meet two requirements:

- *Requirement R1:* Equivalent entities are matched, where the ground truth is the set of pairwise entity matches between equivalent entities.

- *Requirement R2:* Equivalent attributes are matched, where the ground truth is the collection of pairwise attribute matches between equivalent attributes.

Experiment 2: *Identifying when the same entity occurs in negative scenarios.* This experiment involves *Scenario Sets 9* to *16* in Fig. 3, and reports on the ability of matching systems in scenarios where no correspondences exist:

- *Requirement R1:* Different entities are not matched, where the ground truth is that there are no pairwise entity matches between different entities.
- *Requirement R2:* Different attributes are not matched, where the ground truth is that there are no pairwise attribute matches between pairs of attributes.

Experiment 3: *Identifying many-to-one attribute correspondences in positive scenarios.* This experiment involves *Scenario Sets 17* and *18* in Fig. 4, where each element in the ground truth is a collection of attribute matches between each attribute in the set and the single attribute.

Experiment 4: *Identifying many-to-many entity correspondences in positive scenarios.* This experiment involves *Scenario Sets 19* and *20* in Fig. 5.

- *Requirement R1*: Each entity in the source set should be matched to all entities in the target set. The ground truth is the collection of pairwise entity matches between each entity in the source set and all entities in the target set.

The following two requirements are investigated only when the evaluated systems are able to meet R1.
- *Requirement R2*: Primary key attributes in each entity in the source set should be matched to primary key attributes in all entities in the target set. The ground truth is the collection of pairwise attribute matches between primary key attributes in each entity in the source set and primary key attributes in all entities in the target set.
- *Requirement R3*: Partitions in the source schema are matched against partitions in the target schema, with a view to identifying specific types of many-to-many correspondences. For each type, the ground truth is the collection of pairwise attribute matches between attributes as described below:

– *Horizontal Partitioning vs Horizontal Partitioning:* Each non-key attribute in each entity in the source (target) set should be matched to a single non-key attribute in every entity in the target (source) set.
– *Vertical Partitioning vs Vertical Partitioning:* Each non-key attribute in each entity in the source (target) set should be matched to a single non-key attribute in an entity in the target (source) set.
– *Horizontal Partitioning vs Vertical Partitioning:* Each non-key attribute in each entity in the source set should be matched to a single non-key attribute in an entity in the target set; but each non-key attribute in each entity in the target set should be matched to a single non-key attribute in each entity in the source set.

4 Application of MatchBench

4.1 Matching Systems

In general, we follow the advice of the developers when configuring matching systems, for example, by employing the settings suggested in the published papers or in private communication with the authors. In addition, we take all steps that are available to us in order to help the systems to perform well, e.g., by plugging an instance-level matcher into Similarity Flooding and Harmony, which were both originally supplied with only schema-level matchers.

COMA++ [8] is a schema matching platform that supports the composition of schema and instance level matchers from a substantial library. In particular, we applied *AllContext* as the matching strategy, selected matchers *Name*, *NamePath*, *Leaves* and *Parents* at the schema-level and *Content-based* at the instance-level, and employed *Average* for aggregation, *Both* for direction, *Threshold+MaxDelta* for selection and *Average* for combination, as they are demonstrated to be effective in published experimental evaluations [8]. As experience with COMA++ has not given rise to consistent recommendations for Threshold and Delta [16, 8], we decided to employ the default settings of Threshold and Delta (i.e., 0.1 and 0.01) provided with the COMA++ tool.

Similarity Flooding (SF) [17] is a schema matching operator used by the model management platform, Rondo [18]. SF applies a name matcher *NGram* and a similarity flooding algorithm to generate candidate matches, and selects a best match for each element from the candidate matches under the constraint that each element can only be associated with a single match.

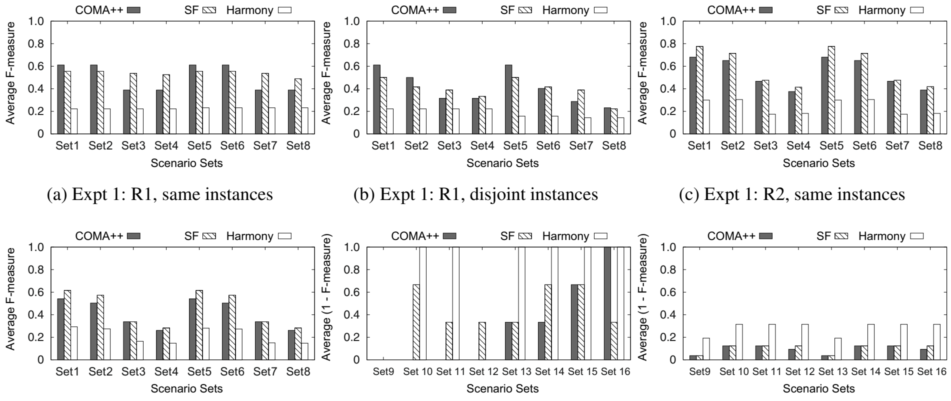
For the evaluation of SF using MatchBench, the *NGram* matcher and an instance matcher (i.e., the *Content-based* matcher of COMA++) are used together to enable SF making use of instance-level information. This improvement turns out to be important for identifying schematic correspondences.

Harmony [22] is an interactive matching tool contained in a suite of data integration tools, called OpenII [21]. For the evaluation using MatchBench, we chose the *EditDistance*, *Documentation* and *Exact* matchers provided by Harmony but we left out the *Mapping* matcher as we do not consider previous matches during the evaluation.

Harmony returns all candidate matches and allows the user to slide a threshold bar while visually observing which matches pass different thresholds. However, there are a large number of scenarios in MatchBench, thus selecting a threshold manually for each of them is not practical. Therefore, we decided to follow the recommendation of the OpenII authors. We use the top matches associated with each element while not restricting the number of matches associated with an element. In addition, as Harmony only works at the schema-level, we combine it with the *Content-based* matcher of COMA++, to provide the same basis in terms of instance-based matches as COMA++ and SF.

4.2 Effectiveness Comparison

Experiment 1: *Identifying when the same entity occurs in positive scenarios.* The results of this experiment are presented in Fig. 6(a) to (d). The following can be observed: (i) All three systems have been reasonably successful at matching equivalent entities and equivalent attributes when they have the same instances (*recalls* reported are fairly high, though not illustrated here), but have been less successful for disjoint instances.



(a) Expt 1: R1, same instances

(b) Expt 1: R1, disjoint instances

(c) Expt 1: R2, same instances

(d) Expt 1: R2, disjoint instances

(e) Expt 2: R1

(f) Expt 2: R2

Fig. 6. Experiments 1 and 2 for COMA++, Similarity Flooding (SF) and Harmony

(ii) A significant number of false positives between different entities and between different attributes have been generated by all systems (the *F-measures* reported in Fig. 6(a) to (d) are fairly low, given high *recalls*). This is due to the selection strategies these

platforms employ: for COMA++, the *MaxDelta* method always chooses a few of the top matches associated with an element, even though the scores of these matches may be fairly low due to the low threshold of 0.1; SF only returns 1-to-1 matches by selecting a best match for each element, regardless of its similarity score; and Harmony keeps a match as long as it is the top match for either of its associated elements irrespective of the match scores, resulting in a large number of incorrect matches, which makes it perform worst among the three platforms.
(iii) Changing the names of equivalent entities into similar or different has almost no impact on the three platforms on matching equivalent attributes (Fig. 6(c) and (d)).

Experiment 2: *Identifying when the same entity occurs in negative scenarios.* The results of this experiment are presented in Fig. 6(e) and (f). The following can be observed: (i) All three systems have matched the two different entities when similarities have been injected into their names or their attributes (Average(1 – *F-measure*) $>$ 0 in Sets *13* to *16* in Fig. 6(e)). This is because all three systems choose the top candidate matches for each element, and this also indicates that entities are matched because they are more similar to each other than to other entities, but not because they represent the same real world notion. (ii) COMA++ and SF perform satisfactorily in not matching different attributes (Fig. 6(f)). Where attributes are matched, this is normally because similar attributes have been introduced, and the remainder results from overlaps in the instances or the names of non-equivalent attributes. Harmony has matched several different attributes even in the baseline scenarios where no similarities have been injected. This shows that its selection strategy that keeps a top match for each element is not effective in deciding the final matches.

Experiment 3: *Identifying many-to-one attribute correspondences in positive scenarios.* The results of this experiment are presented in Fig. 7. COMA++ and SF have failed in this experiment. In contrast to SF, which only identifies 1-to-1 matches, the *Threshold+MaxDelta* method COMA++ uses allows the identification of n-to-1 matches. However, given the *delta* value of 0.01, the *MaxDelta* method sets a fairly small tolerance range below the top match of an attribute, thus only being able to return matches whose similarities are close to the top match. Harmony has identified some n-to-1 attribute correspondences, where the *n* attributes and the *1* attribute have similar names (Sets *17 SI* and *17 DI* in Fig. 7), because Harmony chooses a best match for each element and allows a match to be kept as long as it is the best match for either of its associated elements. When the *n* attributes and the *1* attribute have similar names, the matches between the *n* attributes and the *1* attribute are usually the top matches for the *n* attributes, and thus are selected by Harmony.

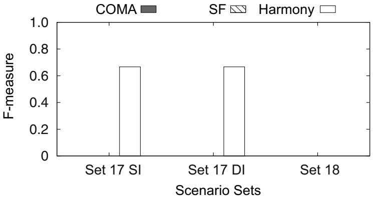
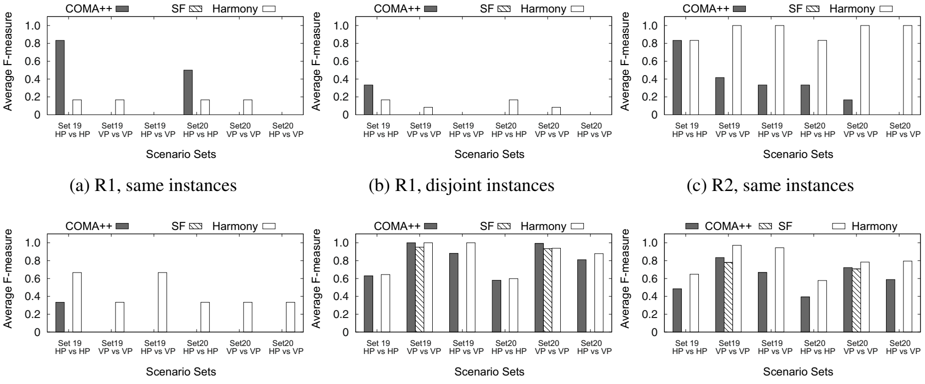


Fig. 7. Experiment 3 for COMA++, Similarity Flooding (SF) and Harmony

Nevertheless, an n-to-1 attribute correspondence refers to a transformation of instances (e.g., string concatenation or numeric operation) between the *n* attributes and the *1* attribute rather than a selection of matches whose similarities are close or the top, as determined by comparing names or instances of pairwise attributes. We anticipate that iMAP [7] could identify the n-to-1 attribute correspondences, however, the system is not publicly available.

Experiment 4: *Identifying many-to-many entity correspondences in positive scenarios.* In this experiment, SF is not able to carry out the task due to its focus on 1-to-1 matches. Where SF identifies a few matches (Fig. 8(e) and (f)), it is because the ground truth is the 1-to-1 matches in vertical partitioning. COMA++ and Harmony have performed rather patchily in seeking to meet requirement *R1*, as presented in Fig. 8(a) and (b), though COMA++ and Harmony have performed satisfactorily on investigating requirements *R2* and *R3*. The following can be observed for COMA++ and Harmony.

(i) COMA++ has only been able to associate the n-to-m entities, i.e., to satisfy requirement *R1*, where the same instances are being represented in the horizontal partitioning models (Set *19 HP vs HP* and Set *20 HP vs HP* in Fig. 8(a)), but has failed in other partitioning models or in disjoint instances. This is because only when the two original entities that have the same instances are horizontally partitioned, the similarities between each pair of entities in the source and target sets are close, and as such are selected by the *MaxDelta* method. Harmony has performed slightly better than the others. However, it has been fairly generous (the *recalls* are always high, but the *precisions* are fairly low). Therefore, the patchy results shown by Harmony are because the equivalent n-to-m entities have also been matched to different entities.



(a) R1, same instances

(b) R1, disjoint instances

(c) R2, same instances

(d) R2, disjoint instances

(e) R3, same instances

(f) R3, disjoint instances

Fig. 8. Experiment 4 for COMA++, Similarity Flooding (SF) and Harmony

(ii) Similar to requirement *R1*, when the alternatively fragmented entities have the same instances and no changes have been made to attribute names, and thus the similarities of matches for many-to-many primary key attributes are close, COMA++ has generally been successful in satisfying requirement *R2*, as shown in Fig. 8(c). Harmony has performed fairly satisfactorily in satisfying requirement *R2* in the SI case, however, for cases where there is less evidence (e.g., the DI case), equivalent primary key attributes are matched to different attributes (Fig. 8(c) and (d)).

(iii) COMA++ has been generally successful at matching non-key attributes, i.e., satisfying requirement *R3*, in both scenario sets where the same instances are represented, but has performed slightly worse in the presence of disjoint instances. COMA++ has performed particularly well in the vertical partitioning scenarios (Set *19 VP vs VP* and Set *20 VP vs VP* in Fig. 8(e)), as the non-key attributes only have single corresponding attributes; but has performed less well in the horizontal partitioning scenarios (Set *19 HP vs HP* and Set *20 HP vs HP* in Fig. 8(e)) where many-to-many correspondences between non-key attributes should be identified. This indicates that COMA++ is more suited to identifying one-to-one correspondences than to many-to-many correspondences. Harmony has been competitive with COMA++ in the SI case, but has performed better in the DI case (Fig. 8(e) and (f)), as the lack of a threshold means that Harmony tends to return more matches, some of which are true positives.

4.3 Summary

The following lessons have been learned from the application of the representative matchers to the benchmark:

(1) The existing schema matching methods were designed principally to associate similar schema elements, and have been shown to perform rather better at this task than at diagnosing the schematic heterogeneities of Kim *et al.* [14].
(2) The existing schema matching methods were more designed for identifying one-to-one matches than for identifying many-to-many schematic correspondences.
(3) The strategy for selecting candidate matches influences the overall effectiveness of schema matching methods significantly.
(4) COMA++ offers alternative choices for different matching tasks. We anticipate that with more appropriate *threshold* and *delta* values, COMA++ would have performed better in experiments provided in MatchBench [15]. However, as a well-known problem, this presents practical challenges that setting any parameters generally requires access to at least some training data.
(5) SF always identifies one-to-one matches between elements of data sources, and thus cannot be used in diagnosing many-to-many schematic heterogeneities.
(6) Designed as an interactive tool, Harmony seems unsuitable for scenarios where a very large number of matching tasks are required and where the automatic generation of matches are demanded, since it is not practical to manually choose matches in such scenarios for individual human users.

5 Related Work

This section reviews work that is related to that carried out here, and considers in particular *experimental evaluation practice for schema matching*, *generation of test cases for schema matching and mapping*, and *existing benchmarks for matching*.

In terms of *experimental evaluation practice for schema matching*, most results have been presented in the context of specific matching proposals, as compared by Do *et al.* [9]. This comparison makes explicit that the diversity in evaluation practice is problematic, thus providing motivation for the development of benchmarks [9]. Overall, the comparison indicates that most evaluations have been carried out using representative real-world schemas; while this approach provides insights into the effectiveness of techniques in specific contexts, the lack of fine-grained control over properties of the matched schemas can make it difficult to understand precisely the circumstances in which methods are effective. Rather than revisiting the ground covered by Do *et al.*, here we focus on the published evaluations of COMA++, SF and Harmony, to which MatchBench is applied in Section 4.

The most comprehensive comparative activity of relevance to MatchBench is the Ontology Alignment Evaluation Initiative (OAEI) [1], which runs an annual event on evaluating ontology matching systems. Whereas, MatchBench is designed to assess whether specific relationships, i.e., schematic correspondences, can be identified by schema matching systems.

In terms of *generation of test cases for schema matching and mapping*, test cases have been generated to support both tuning of matching systems in eTuner [15] and evaluation of schema mapping platforms in STBenchmark [2]. The test schemas over which eTuner is evaluated are generated by applying a number of rules for introducing perturbations into existing schemas. These perturbations overlap with those described in Section 3, but differ in the following respects: (i) they do not follow an established classification of schematic correspondences; (ii) the emphasis is on 1-to-1 matches; (iii) no negative scenarios are described; and (iv) there is no specific identification of collections of test scenarios.

STBenchmark [2] is a benchmark for comparing visual interactive mapping construction systems that aim at assisting an expert in generating a precise specification of mappings between two schemas with less effort. STBenchmark provides rules for generating mapping scenarios and evaluates the degree of effort supported by a schema mapping system in specifying mappings. In essence, these rules overlap with those described in Section 3. However, selecting types of variations and generating evaluation scenarios is the responsibility of users. On the other hand, MatchBench supports the developers of matchers through the provision of immediately usable scenarios.

In terms of *existing benchmarks for matching*, XBenchmark [10] has been developed in the context of XML schema matching, and STBenchmark has been used for schema mapping generation [2]. XBenchmark reports results at a very coarse grain, and is agnostic as to the test cases used. In contrast, we systematically generate test cases to assess the capabilities of matchers in specific scenarios with known properties, and have an overall objective of ascertaining whether or not the matchers provide the diagnostic information required to identify specific schematic heterogeneities. STBenchmark aims for evaluating interactive tools for constructing mappings from matchings, such as Clio [11] or BizTalk Mapper[5], and thus the benchmark measures the amount of human effort involved in addressing specific mapping scenarios given specific matchings. As such, STBenchmark is complementary to MatchBench; indeed, insights from MatchBench may inform the development of helper components for interactive mapping tools that suggest to users what mappings may be most appropriate in a given setting.

[5] www.microsoft.com/biztalk

6 Conclusions

This paper has presented a benchmark for schema matching methods that identifies the extent to which these methods are successful at identifying correspondences between schemas in the presence of the schematic heterogeneities of Kim *et al.* [14]. This is in contrast to most reported evaluations of matching methods, where the focus is on the identification of 1-to-1 matches between individual schema elements, where the ability to combine these observations to draw higher level conclusions has not been investigated.

The objective of the benchmark is not to seek to identify which matching methods are "better" than others, but rather to enhance understanding of when and why specific matching methods are suitable for a given task, with a view to guiding matcher selection and configuration. In providing a revised focus for the evaluation of matching methods, on diagnosing the heterogeneities that mappings must resolve, the benchmark both supports the reassessment of existing proposals and timely evaluation of new techniques.

References

[1] Ontology Alignment Evaluation Initiative (OAEI), `http://oaei.ontologymatching.org/`

[2] Alexe, B., Tan, W.C., Velegrakis, Y.: Stbenchmark: towards a benchmark for mapping systems. PVLDB 1(1), 230–244 (2008)

[3] Bernstein, P., Melnik, S.: Model management 2.0: manipulating richer mappings. ACM SIGMOD, 1–12 (2007)

[4] Bernstein, P.A., Madhavan, J., Rahm, E.: Generic schema matching, ten years later. PVLDB 4(11), 695–701 (2011)

[5] Bonifati, A., Chang, E.Q., Ho, T., Lakshmanan, L.V.S., Pottinger, R., Chung, Y.: Schema mapping and query translation in heterogeneous p2p xml databases. VLDB J. 19(2), 231–256 (2010)

[6] Bonifati, A., Mecca, G., Pappalardo, A., Raunich, S., Summa, G.: Schema mapping verification: the spicy way. In: EDBT, pp. 85–96 (2008)

[7] Dhamankar, R., Lee, Y., Doan, A., Halevy, A.Y., Domingos, P.: imap: Discovering complex mappings between database schemas. In: SIGMOD Conference, pp. 383–394 (2004)

[8] Do, H., Rahm, E.: Matching large schemas: Approaches and evaluation. Information Systems 32(6), 857–885 (2007)

[9] Do, H.-H., Melnik, S., Rahm, E.: Comparison of schema matching evaluations. In: Chaudhri, A.B., Jeckle, M., Rahm, E., Unland, R. (eds.) NODe-WS 2002. LNCS, vol. 2593, pp. 221–237. Springer, Heidelberg (2003)

[10] Duchateau, F., Bellahsene, Z., Hunt, E.: Xbenchmatch: a benchmark for xml schema matching tools. In: VLDB, pp. 1318–1321 (2007)

[11] Fagin, R., Haas, L.M., Hernández, M., Miller, R.J., Popa, L., Velegrakis, Y.: Clio: Schema mapping creation and data exchange. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Conceptual Modeling: Foundations and Applications. LNCS, vol. 5600, pp. 198–236. Springer, Heidelberg (2009)

[12] Franklin, M., Halevy, A., Maier, D.: From databases to dataspaces: a new abstraction for information management. SIGMOD Record 34(4), 27–33 (2005)

[13] Haas, L.: Beauty and the beast: The theory and practice of information integration. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 28–43. Springer, Heidelberg (2006)

[14] Kim, W., Seo, J.: Classifying schematic and data heterogeneity in multidatabase systems. IEEE Computer 24(12), 12–18 (1991)

[15] Lee, Y., Sayyadian, M., Doan, A., Rosenthal, A.: etuner: tuning schema matching software using synthetic scenarios. VLDB J. 16(1), 97–122 (2007)

[16] Massmann, S., Engmann, D., Rahm, E.: Coma++: Results for the ontology alignment contest oaei 2006. Ontology Matching (2006)

[17] Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: a versatile graph matching algorithm and itsapplication to schema matching. In: ICDE, pp. 117–128 (2002)

[18] Melnik, S., Rahm, E., Bernstein, P.: Rondo: a programming platform for generic model management. In: ACM SIGMOD, pp. 193–204 (2003)

[19] Ozsu, M.T., Valduriez, P.: Principles of distributed database systems. Addison-Wesley, Reading Menlo Park (1989)

[20] Rahm, E., Bernstein, P.: A survey of approaches to automatic schema matching. The VLDB Journal The International Journal on Very Large Data Bases 10(4), 334–350 (2001)

[21] Seligman, L., Mork, P., Halevy, A.Y., Smith, K., Carey, M.J., Chen, K., Wolf, C., Madhavan, J., Kannan, A., Burdick, D.: Openii: an open source information integration toolkit. In: SIGMOD Conference, pp. 1057–1060 (2010)

[22] Smith, K., Morse, M., Mork, P., Li, M.H., Rosenthal, A., Allen, D., Seligman, L.: The role of schema matching in large enterprises. In: CIDR (2009)

Towards Performance Evaluation of Semantic Databases Management Systems

Bery Mbaiossoum[1,2], Ladjel Bellatreche[1], and Stéphane Jean[1]

[1] LIAS/ISAE-ENSMA - University of Poitiers 86960, Futuroscope Cedex, France
{mbaiossb,bellatreche,jean}@ensma.fr
[2] University de N'Djamena, Chad Republic

Abstract. The spectacular use of ontologies generates a big amount of semantic instances. To facilitate their management, a new type of databases, called semantic databases ($\mathcal{SDB}$) is launched. Large panoply of these $\mathcal{SDB}$ exists. Three main characteristics may be used to differentiate them: (i) the storage layouts for storing instances and the ontology, (ii) ontology modeling languages, and (iii) the architecture of the target database management system (DBMS) supporting them. During the deployment phase, the database administrator (DBA) is faced to a choice problem (which $\mathcal{SDB}$ she/he needs to choose). In this paper, we first present in details the causes of this diversity. Based on this analysis, a generic formalization of $\mathcal{SDB}$ is given. To facilitate the task of the DBA, mathematical cost models are presented to evaluate the performance of each type of $\mathcal{SDB}$. Finally, two types of intensive experiments are conducted by considering six $\mathcal{SDB}$, both issued from industry and academic communities; one based on our mathematical cost models and another based on the studied semantic DBMS cost models.

Keywords: Semantic Databases, Cost Models, Query Performance.

1 Introduction

The database is one of the robust technologies. Along its history, we remark that when a new data model (e.g., the relational, object or XML data model) is considered; the database technology offers *storage*, *querying* and *management* solutions to deal with these new data. These solutions were directly implemented in academic and industrial DBMS. In the last decades, the semantic technology got a lot of attention from the research community, where semantic (ontological) data models were proposed. This phenomenon generates a large amount of semantic data that require efficient solutions to store and manage them. In order to honour its tradition and to keep its marketplace, the database technology responded to this need and offered *persistent solutions*. As a consequence, a new type of databases is created, called *semantic databases* ($\mathcal{SDB}$), to store both data and ontologies in the same repository. A large panoply of $\mathcal{SDB}$ exists. OntoDB [1], Jena [2] or Sesame [3] are examples of academician $\mathcal{SDB}$. Oracle [4] and IBM [5,6] are example of solutions coming from industry.

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 107–120, 2013.

Contrary to traditional databases, $\mathcal{SDB}$ bring new dimensions: (1) *the diversity of ontology formalisms*: each $\mathcal{SDB}$ uses a particular formalism to define its ontologies (e.g., OWL [7] or PLIB [8]), (2) *the diversity of storage layouts*: in a $\mathcal{SDB}$, several storage layouts (horizontal, vertical, binary) are used to store ontologies and their data, and (3) *the diversity of architectures*: three main architectures of DBMS managing $\mathcal{SDB}$ are distinguished according to the number of schemes used to store ontologies, data and eventually the ontology formalism used to define them. These dimensions complicates the deployment phase of $\mathcal{SDB}$ as each dimension may impact positively or negatively the performance of the target applications. Thus evaluating different $\mathcal{SDB}$ becomes a crucial issue for DBA. To do so, two directions are possible: (i) the use of mathematical cost models and (ii) the real deployment of $\mathcal{SDB}$ on several DBMS. Most of studies have been concentrated on the physical phase of performance of $\mathcal{SDB}$, where algorithms for selecting optimization structures such as materialized views [9] are proposed. These studies assume that a fixed $\mathcal{SDB}$ is used and do not consider the dimensions that we discussed. Thus the proposed cost models do not cover all possible combinations of deployment, they only consider one $\mathcal{SDB}$ with a fixed ontology formalism model, storage layout and architecture.

In this paper, we propose a formalization of the notion of $\mathcal{SDB}$. The different components of this formal model are illustrated. This formalization allows us to compare the existing $\mathcal{SDB}$. Cost models depending on the different types of $\mathcal{SDB}$ are proposed and used to compare the existing $\mathcal{SDB}$. To the best of our knowledge, our proposal is the first one dealing with the development of cost models considering the large diversity of $\mathcal{SDB}$. To validate this cost model, several experiments are run on the LUBM benchmark.

This paper consists of six sections. Section 2 defines basic notions related to ontologies and presents the diversity of $\mathcal{SDB}$. Section 3 introduces a formalization of $\mathcal{SDB}$ and the section 4 defines our cost model. We present in section 5 the performance evaluations of several $\mathcal{SDB}$. Finally, section 6 concludes the paper.

2 Background: The Diversity of $\mathcal{SDB}$

A variety of $\mathcal{SDB}$ have been proposed in the last decade. Some $\mathcal{SDB}$ only consider the management of ontology instances represented as RDF data while others also support the management of ontologies and ontology formalisms inside the database. Moreover the management of these data is based on different storage layouts. In this section, we introduce basic notions about ontology and detail the different storage layouts and architectures of $\mathcal{SDB}$.

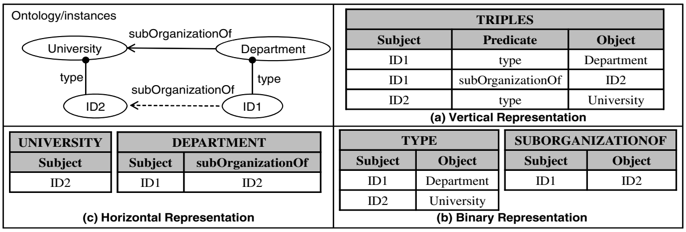
2.1 Ontology Definition and Formalism

An ontology is a consensual model defined to explicit the semantics of a domain by a set of concepts (class or property) that can be referenced by universal identifiers (e.g., URI). Two main types of concepts in a conceptual ontology are distinguished: *primitive* and *defined concepts*. *Primitive concepts* (or canonical concepts) represent concepts that can not be defined by a complete axiomatic definition. They define the border of the domain conceptualized by an ontology. *Defined concepts* (or non canonical concepts) are defined by a complete axiomatic definition expressed in terms of other concepts (either primitive or defined concepts). These concepts are the basis of inference mechanisms like automatic classification.

Several formalisms (or languages) have been proposed for defining ontologies. They differ on their descriptive and deductive capabilities. Some ontology languages focuses mainly on the definition of primitive concepts. Thus they are used to define ontologies that are consensual and enhanced conceptual models. RDF Schema and PLIB are two examples of such languages. RDF Schema is a language defined for the Semantic Web. It extends the RDF model to support the definition of classes and properties. The PLIB (Parts Library) formalism is specialized in the definition of ontology for the engineering domain which often requires a precise description (e.g., value scaling and context explication) of primitive concepts. OWL Lite, DL and Full are ontology models with more deductive capabilities. They support the definition of defined concepts with various constructors such as restrictions (e.g, the definition of the *Man* class as all persons whose gender is male) or Boolean expression (e.g., the definition of the *Human* class as the union of the *Man* and *Woman* classes).

2.2 Storage Layouts Used in $\mathcal{SDB}$

Three main storage layouts are used for representing ontologies in databases [10]: vertical, binary and horizontal. These storage layouts are detailed below and illustrated on a subset of the LUBM ontology in the Figure 1.



TRIPLES		
Subject	Predicate	Object
ID1	type	Department
ID1	subOrganizationOf	ID2
ID2	type	University

(a) Vertical Representation

UNIVERSITY
Subject
ID2

DEPARTMENT	
Subject	subOrganizationOf
ID1	ID2

(c) Horizontal Representation

TYPE	
Subject	Object
ID1	Department
ID2	University

SUBORGANIZATIONOF	
Subject	Object
ID1	ID2

(b) Binary Representation

Fig. 1. Main storage layouts used by $\mathcal{SDB}$

- *vertical storage layout:* it consists of a single triples table with three columns `(subject, predicate, object)`. Since URI are long strings, additional tables may be used to store only integer identifier in the triple table. Thus this storage layout is a direct translation of the RDF model. It can be used to store the different abstraction layers related to the management of ontologies. The main drawback of this storage layout is that most queries require a lot of self-join operations on the triple table [10].The $\mathcal{SDB}$ Sesame (one version of it) and Oracle use this storage layout.

- *binary storage layout:* it consists of decomposing the triple table into a set of 2-columns tables `(subject, object)`, one for each predicate. In some implementations, the inheritance of classes and class membership are represented in a different way (e.g., the class membership can be represented by a unary table for each class or the inheritance using the table inheritance of PostgreSQL). Compared to the vertical storage layout, this storage layout results in smaller tables but queries can still require many joins for queries involving many properties [1]. The $\mathcal{SDB}$ SOR uses the binary storage layout for the representation of ontologies and their instances.
- *horizontal storage layout:* it consists of a set of usual relational tables. For storing ontologies, this storage layout consists of a relational schema defined according to the ontology formalism supported. For managing ontology instances, a table $C(p_1, \ldots, p_n)$ is created for each class `C` where $p_1, \ldots, p_n$ are the set of single-valued properties used at least by one instance of the class. Multi-valued properties are represented by a two-column table such as in the binary representation or by using the array datatype available in relational-object DBMS. Since all instances do not necessarily have a value for all properties of the table, this representation can be sparse which can impose performance overhead. The $\mathcal{SDB}$ OntoDB uses the horizontal storage layout for storing instances, ontologies and the ontology formalism.

2.3 Architectures of $\mathcal{SDB}$

According to the abstraction layers managed (instances, ontologies and ontology formalism) and the storage layouts used, $\mathcal{SDB}$ are decomposed into one or several schemes leading to different architectures illustrated Figure 2.

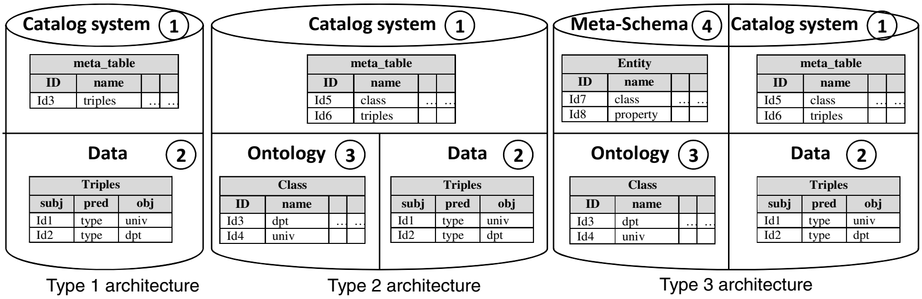


Fig. 2. Different Architectures of $\mathcal{SDB}$

- *Type 1 architecture:* some $\mathcal{SDB}$ like Oracle or Jena use only one schema to store all information. As a consequence, these $\mathcal{SDB}$ have two parts like classical database: the data and system catalog parts.
- *Type 2 architecture:* other $\mathcal{SDB}$ like IBM SOR [5] have chosen to separate the storage of ontology instances from the storage of the ontology (classes and properties) in two different schemes. This separation leads to $\mathcal{SDB}$ composed of three parts: data, ontology and system catalog.

- *Type 3 architecture:* $\mathcal{SDB}$ like OntoDB [1] have introduced a fourth part to store the ontology formalism used. This part, called the *metaschema* in OntoDB, is a specialized system catalog for the ontology part. With this part, users can modify the ontology formalism used in the $\mathcal{SDB}$.

3 Formalization of $\mathcal{SDB}$ and Comparison Features

3.1 Formalization of $\mathcal{SDB}$

In the previous sections we have seen that $\mathcal{SDB}$ presents an important diversity in terms of architecture, storage layouts and ontology formalisms supported. To provide a general view of $\mathcal{SDB}$ that can be used as a basis for the physical design of $\mathcal{SDB}$, we propose the following formalization:
$\mathcal{SDB}$: $< MO, I, Sch, Pop, SM_{MO}, SM_{Inst}, Ar >$, where :

- MO represents the ontology (model part). It is formalized by the following 5-tuple: $< C, P, Applic, Ref, Formalism >$ where :
  - C represents the classes of the ontology.
  - P represents the properties of the ontology.
  - $Applic : C \rightarrow P^2$ is a function returning all properties whose domain is a given class.
  - $Ref : C \rightarrow (operator, exp(C))$ is a function that maps each class to an operator (inclusion or equivalence) and an expression of other classes. It is used to represent the relationship between classes (subsumption, Boolean operators, etc.).
  - Formalism is the ontology formalism used to define the ontology.

  For example, an OWL ontology is defined by: $< Classes, Properties, Applic, descriptionlogicoperators, OWL>$ where $descriptionlogicoperators$ is the set of description logic operators supported by the given version of OWL.
- I: represents the set of ontology instances.
- $Sch : C \rightarrow P^2$ is a function that maps each class to the set of properties valued by at least one instance of this class.
- $Pop : E \rightarrow 2^I$ is a function that associates a class to its instances.
- SM_{MO} is the storage layout used to represent the ontology (vertical, horizontal or binary).
- SM_{Inst} is the storage layout used to represent the ontology instances.
- Ar is the $\mathcal{SDB}$ architecture ($Type_1$, $Type_2$, $Type_3$).

According to this formalization, the $\mathcal{SDB}$ Oracle is represented by:
$SDB_{Oracle} :< MO :<Classes, Properties, Applic, Operators\,(RDFS, OWLSIF$ or $OWLPrime), (RDFS$ or $OWL)>, RDFInstances, \phi, tablesRDF\_link$ and RDF_values giving instances of each class, $Vertical, Vertical, Type_1>$
This model gives a general view on the diversity of $\mathcal{SDB}$. In the next section, we gives a more precise comparison of $\mathcal{SDB}$ by defining some key futures of these systems.

3.2 Key Features of $\mathcal{SDB}$

To deepen our study, we have compared six $\mathcal{SDB}$: three coming from research (OntoDB [1], Sesame [3] and Jena [2]) and three coming from industry (Oracle [4], DB2RDF [6] and IBM SOR [5]). From our study, we have identified a set of key features of $\mathcal{SDB}$. These features include :

– *Natural language support:* this feature consists in using the linguistic information (sometimes in several natural languages) available in an ontology.
– *Query language:* the query language supported by the $\mathcal{SDB}$.
– *Version and evolution of ontologies:* this feature consists in providing a way of modifying an ontology while keeping track of the modification done.
– *Inference support:* this feature consists in supporting the inference rule defined for the ontology formalism supported.
– *User-defined rules:* this feature consists in allowing users to define their own derivation rules.

Table 1 shows these key features of $\mathcal{SDB}$ (in addition to the previous identified criteria) and their availability in the $\mathcal{SDB}$ studied.

Table 1. Comparative study of $\mathcal{SDB}$ (V : vertical, H: horizontal, B: binary, H: hybrid)

Features	**Oracle**	**SOR**	**OntoDb**	**Jena**	**Sesame**	**Db2rdf**
Formalism supported	RDF, OWL	OWL	PLIB	RDF, OWL	RDF, OWL	RDF
Natural Language Support	yes	yes	yes	no	no	no
Query Languages	sql, sparql	sparql	ontoql, sparql	rql, Rdql, sparql	serql, sparql	sql, sparql
Version and evolution	no	no	yes	no	no	no
Inference support	yes	yes	no	yes	yes	no
User-defined rules	yes	no	no	yes	yes	no
Ontology Storage layout	V	H	H	V, H	V, H	V
instances Storage layout	V	B	H	H	V, B	V
Architecture	$Type_1$	$Type_2$	$Type_3$	$Type_1$	$Type_1$	$Type_1$
Underlying DBMS	Oracle	DB2, Derby	Postgres	Oracle, Postgres, Mysql	Oracle, Postgres, Mysql	DB2

The performance of current $\mathcal{SDB}$ is also an important feature. We propose a theoretical comparison through a cost model (Section 4) and an empirical comparison of six $\mathcal{SDB}$ to validate our cost model (section 5).

4 Cost Model

The cost model is an important component of a query optimizer. It can be used for important tasks such as selecting the best query plan or using adaptive optimization techniques. In this section we propose a cost model for $\mathcal{SDB}$ that takes into account their diversity.

4.1 Assumptions

Following assumptions of classical cost models such as those made in System R, we assume that (1) computing costs are lower than disk access costs, (2) statistics about the ontology and their instances are available (e.g., the number of instances by class) and (3) the distribution of values is uniform and attributes are independent of each other.

We also need to make specific assumption for $\mathcal{SDB}$. Indeed logical inference plays an important role in $\mathcal{SDB}$. A $\mathcal{SDB}$ is said to be *saturated* if it contains initial instances and inferred instances, otherwise it is called *unsaturated*. Some $\mathcal{SDB}$ are saturated during the data loading phase either automatically (e.g., IMB SOR) or on demand (e.g., Jena or Sesame). Other $\mathcal{SDB}$ (e.g., Oracle) can be saturated at any time on demand. Our cost function relies on the saturated $\mathcal{SDB}$ (it does not take into account the cost of logical inference).

4.2 Parameters of the Cost Function

As in traditional databases, the main parameters to be taken into account in the cost function of $\mathcal{SDB}$ are: the cost of disk access, the cost of storing intermediate files and the computing cost. Let *q* be a query and *sdb* a semantic database against which *q* will be executed and *cost* the cost function. In terms of architecture model, we can see that compared to a traditional cost model, the cost model in $\mathcal{SDB}$ is *increased*, resulting from the access cost to different parts of the architecture:

$$\begin{aligned}
&Type1 : cost(q, sdb) = cost(syscatalog) + cost(data)\\
&Type2 : cost(q, sdb) = cost(syscatalog) + cost(data) + cost(ontology)\\
&Type3 : cost(q, sdb) = cost(syscatalog) + cost(data) + cost(ontology)\\
&\qquad\qquad\qquad\qquad\qquad\qquad + cost(ontology\ meta - schema)
\end{aligned}$$

The cost to access the system catalog ($cost(syscatalog)$) is part of the cost model of classical databases. It is considered negligible because the system catalog can be kept in memory. Thus, the query cost function depends on the architecture and the storage model of $\mathcal{SDB}$.

We assume that the meta-schema and the ontology-schema are small enough to be also placed in memory. Hence in all architectures the cost can be reduced to the cost of data access, expressed as the number of inputs/outputs ($Cost(q, sdb) = cost(data)$). The cost of queries execution is heavily influenced by the operations done in the query, which are mainly projection, selection and join. Our cost function focuses on these three operations.

4.3 Our Queries Template

We consider queries expressed according to template below. These queries can be expressed in most semantic query languages like SPARQL, OntoQL, etc. If C_1, ..., C_n are ontology classes and p_{11}, ..., p_{nn} properties, the considered query pattern is the following:

$(?id_1, type, C_1)\ (?id_1, p_{11}, ?val_{11})\ \cdots\ (?id_1, p_{n1}, ?val_{n1})\ [FILTER()]$
$(?id_2, type, C_2)\ (?id_2, p_{12}, ?val_{12})\ \cdots\ (?id_2, p_{n2}, ?val_{n2})\ [FILTER()]$
...
$(?id_n, type, C_n)\ (?id_n, p_{1n}, ?val_{1n})\ \cdots\ (?id_n, p_{nn}, ?val_{nn})\ [FILTER()]$

4.4 Cost of Selections and Projections

In our template, a selection is a query that involves a single class. It has the following form: $(?id, type, C)(?id, p_1, ?val_1)\ldots(?id, p_n, ?val_n)[FILTER()]$. We distinguish *single-triple* selections, which are queries consisting of a single triple pattern, and *multi-triples* selections, consisting of more than one triple pattern (all concerning a single class). In the vertical and binary representations only *single-triple* selections are interpreted as selections, because *multi-triples* selections involve joins. We define the cost function as in relational databases. For *single-triple* selection, our function is equal to the number of pages of the table involved in the query:

- vertical layout: $Cost\ (q, sdb) = |T|$, where $|T|$ is the number of pages of the table T. If an index is defined on the triples table, $cost(q, sdb) = P(index) + sel(t) * |T|$, where *P(index)* is the cost of index scanning and *sel(t)* is the selectivity of the triple pattern t as defined in [11].
- binary layout: the selection is done on the *property tables*. $Cost(q, sdb) = |Tp|$ where Tp is the *property table* of the property of the query triple pattern. With an index on the selection predicate, $cost(q, sdb) = P(index) + sel * |Tp|$, where sel is the selectivity of the index.
- horizontal layout: the selection targets the tables of classes domain of the property of the query triple pattern. $Cost(q, sdb) = \sum_{Tcp \in dom(p)}(|Tcp|)$, where Tcp are the tables corresponding to the classes domain of the property of the query triple pattern. If there is an index defined on the selection predicate, $cost(q, sdb) = \sum_{Tcp \in dom(p)}(P(index) + sel * |Tcp|)$ where sel is the index selectivity.

Multi-triples selection queries are translated into joins in vertical and binary layouts. In the horizontal layout, the cost function is the same as the one defined for *single-triple* selection. A projection is a free filter selection having a list of attributes whose size is less than or equal to the number of properties of the class on which it is defined. Its cost is defined in the same way as the selection.

4.5 Cost of Joins

Join operations are done in queries that involve at least two triple patterns. In the horizontal layout these triple patterns must belong to at least two classes from different hierarchies. In the vertical layout, we note that self-join of triples table can be done in two ways: (1) a naive join of two tables *i.e.,* a Cartesian product followed by a selection. We call this a *classic join.* (2) a selection for each triple pattern followed by joins on selection results. We call that a *delayed* *join*. We consider separately these two approaches of self-join of triples table. Our cost function for other approaches is the same as in relational databases, and depends on the join algorithm. Only the cost of hash join is presented in this paper (Tab. 2 where V, B and H represent respectively the Vertical, Binary and Horizontal layouts).

Table 2. Cost of joins (dom(p) : domain of p)

Hash join index	*delayed join*	*classic join*
V: join(T,T), T triples table	$2 * \|T\| + 4 * (\|t1\| + \|t2\|)$	$6 * \|T\|$
B: join(T_1, T_2), T_i Tables of prop.	not applicable	$3 * (\|T_1\| + \|T_2\|)$
H: join(T_1, T_2), T_i tables of classes	not applicable	$\sum_{T \in dom(p)} 3(\|T_1\| + \|T_2\|)$

4.6 Application of Cost Model

To illustrate our cost function, we considered queries 2, 4 and 6 of the LUBM benchmark. We have translated these queries according to the different storage layouts. Then we have made an evaluation of their cost using statistics of the Lubm01 dataset (given in the section 5). Figure 3 presents the results obtained. It shows that processing a self-join query on the triples table with a *classic join* requires more I/O than with a *delayed join*. We observe that for a single-triple selection, the binary layout requires less I/O, so it is likely to be more efficient than other storage layouts. In other types of queries (join queries and multi-triples selections), the horizontal layout provides the best results. This theoretical results have been partially confirmed by our experimental results presented in the next section. Indeed as we will see, OntoDB (horizontal layout) provide the best query response time followed by Oracle (vertical layout). However for some $\mathcal{SDB}$ such as Jena, Sesame and DB2RDF, the evaluation results do not clearly confirm these theoretical results. We believe this is due to the specific optimization done in these systems.

5 Performance Evaluation

We have run experiments to evaluate the data loading and query response time of the six considered $\mathcal{SDB}$. As OntoDB was originally designed for the PLIB formalism we have developed a module to load OWL ontologies.

5.1 Dataset Used in Experiments

We used the benchmark of Lehigh University (denoted LUBM) to generate five datasets with respectively 1, 5, 10, 50 and 100 universities (denoted respectively Lubm01, Lubm05, Lubm10, Lubm50 and Lubm100). The number of instances and triples generated are presented in Table 3. Our experiments were conducted on a 3.10 GHZ Intel Xeon DELL personal computer with 4GB of RAM and 500GB of hard disk. We have used the following DBMS: Oracle 11g, IBM DB2 9.7 for SOR and DB2RDF and PosgresSQL 8.2 for OntoDB, Jena and Sesame.

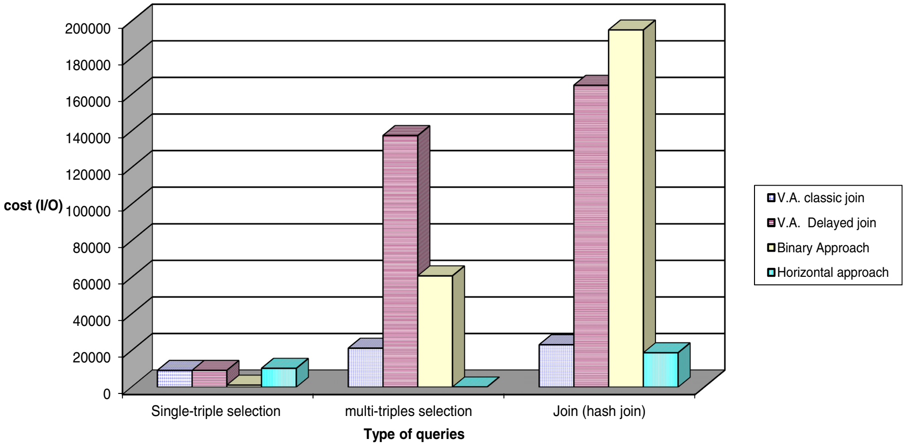


Fig. 3. Queries costs provided by our cost model (V.A. : Vertical approach)

Table 3. Generated datasets

Dataset	Lubm01	Lubm05	Lubm10	Lubm50	Lubm100
# instances	82415	516116	1052895	5507426	11096694
# triples	100.851	625.103	1.273.108	6654856	13405675

5.2 Performance in Terms of Data Loading Time

We loaded and measured the loading time four times for each dataset on each $\mathcal{SDB}$ and took the average time. Results are given in Table 4. Since Sesame can be used as a $\mathcal{SDB}$ with vertical or binary approach, we tested these two approaches. In the following, we call *SesameSdbI* and *SesameSdbII* the Sesame system implemented respectively with the vertical and binary layout. For Oracle we used the rule base *owlprime*.

Table 4. Summary of loading time (in sec)

Dataset	Lubm01	Lubm05	Lubm10	Lubm50	Lubm100
Oracle	12	55	116	1562	39216
DB2RDF	11	53	109	2322	22605
Jena	25	188	558	40862	147109
SesameSdbI	16	158	391	23424	65127
SesameSdbII	27	231	522	33521	260724
OntoDB	15975	72699	146023	–	–
IBM SOR	90	590	1147	–	–

Interpretation of Results. In terms of loading time, as we can see in Table 4, Oracle and DB2RDF provide the best results. This can be explained by the specific and optimized loading mechanisms that have been build on the Oracle and DB2 DBMS. Compared to DB2RDF, SOR is slower as it takes a lot of time to make the inferences on the ontology and instances and we were not able to load LUBM50 and LUBM100. For the other $\mathcal{SDB}$, the results of the two versions of Sesame and Jena are similar with a slight advantage for the vertical layout of Sesame (the triple table which is a direct translation of the RDF model). OntoDB is slower that the other $\mathcal{SDB}$. This overhead is most probably due to our added loading module that imposes a new layer on top of the loading engine. Like with SOR, we were not able to load LUBM50 and LUBM100.

5.3 Performance in Terms of Queries Processing Time

We measured the query response time of queries on each $\mathcal{SDB}$ four times and we kept the average time. We used the 14 queries of the LUBM benchmark, adjusted to be accepted by our $\mathcal{SDB}$. The datasets used are Lubm01 and Lubm100. The results obtained led to the histograms presented Figure 4 and Figure 5. For readability we present the results of the queries that have a long response time in Table 5. As we were not able to load LUBM100 on OntoDB and SOR, the query response times for these dataset are not shown.

Table 5. Query response time on LUBM100 (Q2,Q6,Q7,Q9,Q14) (in sec)

	Q2	Q6	Q7	Q9	Q14
Oracle	240,75	1743	125,76	1973,32	473,23
DB2RDF	95552,5	12815,5	330454,67	692432	6477
Jena	372760,33	147696	224287,33	3538480	163798
SesameSDB1	32920,66	29922,66	21907,66	110498	16707,33
SesameSDB2	115819	70458	2266	242991	64107

Interpretation of Results. Regarding the queries response times, OntoDB reacts better than the other $\mathcal{SDB}$ for most queries executed on LUBM01. Indeed, since OntoDB uses the horizontal layout and that queries involve several property, it performs less join operations than other $\mathcal{SDB}$. Indeed all queries on properties of a same class can be reduced to selection operations on table corresponding to this class, which is not the case when we use an other layout. Query 4 of the benchmark LUBM is a good illustration. This query is made of five triples patterns having all the same domain (*Professor*). This query does not need a join operation in OntoDB, but requires at least four joins in other systems. If the query involves less property (e.g., query 6 of LUBM is a simple selection query on a single property), the query response time is close to the $\mathcal{SDB}$ that use a binary layout as this layout also requires a single scan of a unique property table. For this query, the horizontal layout is the worse as it requires a scan of the whole triple table (Oracle, DB2RDF). We notice that even if Oracle uses an

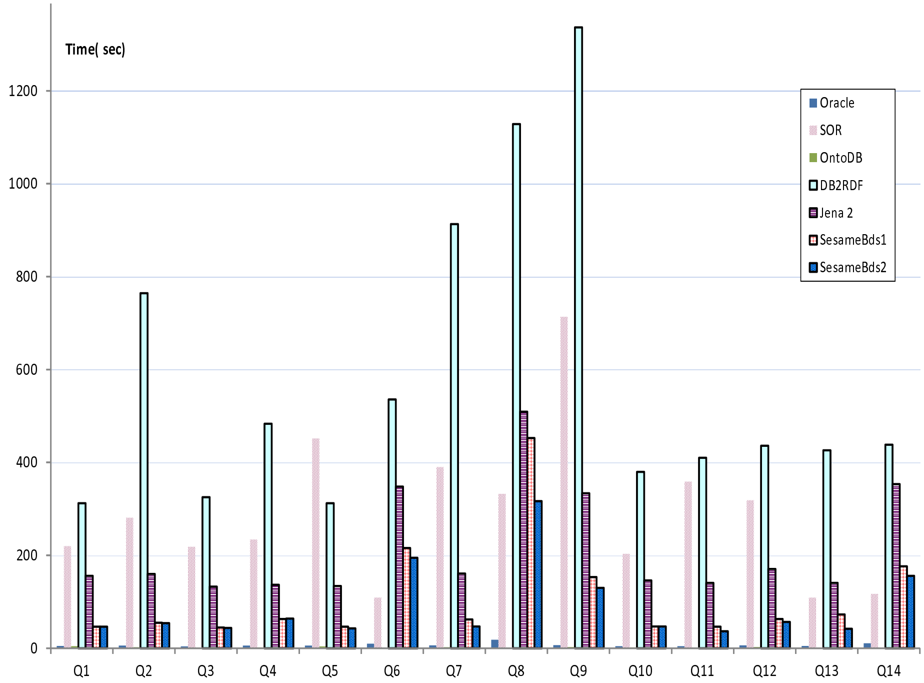


Fig. 4. Queries response time on LUBM01

horizontal layout, the query response time are really close to OntoDB and better than the other $\mathcal{SDB}$. These good results are certainly due to the specific optimization techniques set up in this system. Indeed, several materialized view are used to reduce the need to scan the whole triple table. Considering the Sesame $\mathcal{SDB}$, we note that the binary layout implementation of Sesame (sesameBdsII) outperforms slightly the vertical layout (sesameBdsI). For the $\mathcal{SDB}$ based on DB2, the poor performance of DB2RDF can be explained by the fact that it uses a horizontal layout with a lot of tables linked to the triples table (and so need many joins in addition to the self-joins of the triple table). The result of SOR for LUBM01 are slightly better than DB2RDF but worst than the other $\mathcal{SDB}$. Like for the loading part, this result is due to inference done by SOR during the query processing. Indeed for the LUBM query 12, other $\mathcal{SDB}$ return an empty result since there is no explicit statement of an instance of the *Chair* class. On the contrary SOR returns several results thanks to a deductive process (an instance of *Professor* is also an instance of *Chair* if it is at the *headOf* a *Department*). If we add a rule reflecting this assertion to other $\mathcal{SDB}$ having an inference engine they would also provide answers (but with a worse execution time). Considering the deductive process, Jena and Sesame use inference engines based on a central memory management or file system, but they do not work on data stored in databases yet. It is also possible to use an inference engine during

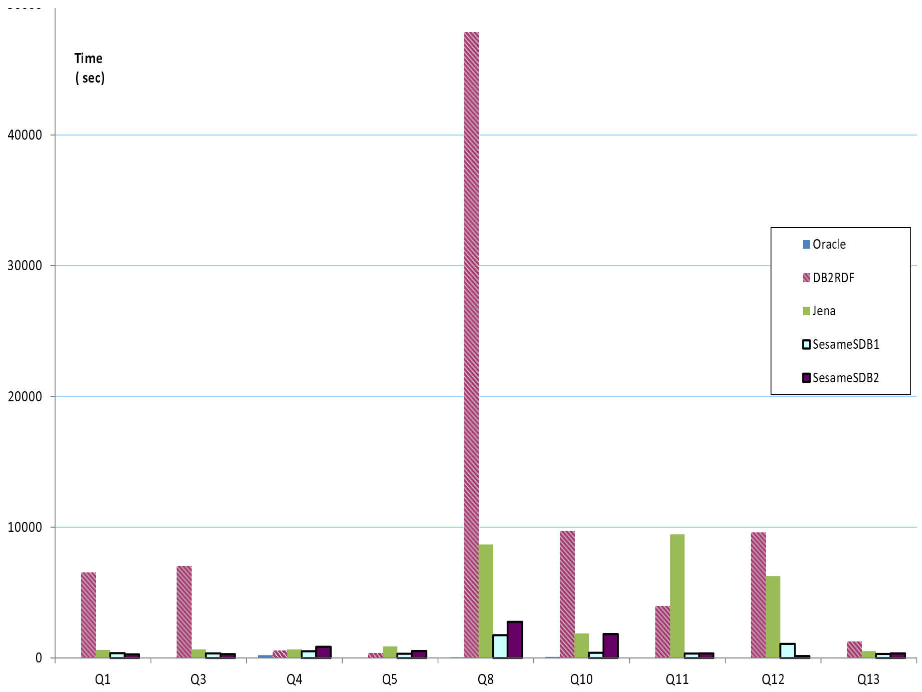


Fig. 5. Queries response time on LUBM100 (Q1,Q3,Q4,Q5,Q8,Q10,Q11,Q12,Q13)

the data loading phase and to store all inferred data in the database. But if we do that the loading time will be worst.

6 Conclusion

In recent years, ontologies have been increasingly used in various domains. Therefore a strong need to manage these ontologies in databases has been felt. As a consequence, both academics and industrialists have proposed persistence solutions based on existing DBMS. Unlike traditional DBMS, $\mathcal{SDB}$ are diverse in terms of ontology formalisms supported, storage layouts and architectures used. To facilitate the understanding of this diversity, we have studied six $\mathcal{SDB}$ and proposed a model that captures this diversity. Considering the performance of $\mathcal{SDB}$, we have conducted a study both theoretically by the definition of a cost model and empirically by measuring the data loading time and query processing time on the LUBM benchmark. The results show that our cost model predict the performance obtained for the $\mathcal{SDB}$ that do not use specific optimizations. Regarding the performances, we first note the effectiveness of industrial semantic databases in terms of data loading time. For the query response time, the results are different. The $\mathcal{SDB}$ that uses an horizontal layout give good results for most queries but the completeness of the inference process has to be taken into account. As a further step in our study of $\mathcal{SDB}$, we plan to modify the data and queries used in our experiment to determine under what conditions a storage layout and/or an architecture is better than an other. The application of our cost model for specific query optimization problem in $\mathcal{SDB}$ (e.g. materialized view selection) is also an important perspective for future work.

References

1. Dehainsala, H., Pierra, G., Bellatreche, L.: Ontodb: An ontology-based database for data intensive applications. In: Kotagiri, R., Radha Krishna, P., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 497–508. Springer, Heidelberg (2007)
2. Wilkinson, K., Sayers, C., Kuno, H., Reynolds, D.: Efficient rdf storage and retrieval in jena2. HP Laboratories Technical Report HPL-2003-266, 131–150 (2003)
3. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 54–68. Springer, Heidelberg (2002)
4. Wu, Z., Eadon, G., Das, S., Chong, E.I., Kolovski, V., Annamalai, M., Srinivasan, J.: Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle. In: Proceedings of the 24th International Conference on Data Engineering (ICDE 2008), pp. 1239–1248 (2008)
5. Lu, J., Ma, L., Zhang, L., Brunner, J.S., Wang, C., Pan, Y., Yu, Y.: Sor: a practical system for ontology storage, reasoning and search. In: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB 2007), pp. 1402–1405 (2007)
6. IBM: Rdf application development for ibm data servers (2012)
7. Dean, M., Schreiber, G.: OWL Web Ontology Language Reference. World Wide Web Consortium (2004), http://www.w3.org/TR/owl-ref
8. Pierra, G.: Context representation in domain ontologies and its use for semantic integration of data. Journal of Data Semantics (JoDS) 10, 174–211 (2008)
9. Goasdoué, F., Karanasos, K., Leblay, J., Manolescu, I.: View Selection in Semantic Web Databases. Proceedings of the VLDB Endowment 5(2), 97–108 (2011)
10. Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: Scalable Semantic Web Data Management Using Vertical Partitioning. In: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB 2007), pp. 411–422 (2007)
11. Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D.: Sparql basic graph pattern optimization using selectivity estimation. In: Proceedings of the 17th International Conference on World Wide Web (WWW 2008), pp. 595–604 (2008)

On the Connections between Relational and XML Probabilistic Data Models

Antoine Amarilli[1] and Pierre Senellart[2]

[1] École normale supérieure, Paris, France
& Tel Aviv University, Tel Aviv, Israel
antoine.amarilli@ens.fr
http://a3nm.net/

[2] Institut Mines–Télécom, Télécom ParisTech, CNRS LTCI, Paris, France
& The University of Hong Kong, Hong Kong
pierre.senellart@telecom-paristech.fr
http://pierre.senellart.com/

Abstract. A number of uncertain data models have been proposed, based on the notion of compact representations of probability distributions over possible worlds. In probabilistic relational models, tuples are annotated with probabilities or formulae over Boolean random variables. In probabilistic XML models, XML trees are augmented with nodes that specify probability distributions over their children. Both kinds of models have been extensively studied, with respect to their expressive power, compactness, and query efficiency, among other things. Probabilistic database systems have also been implemented, in both relational and XML settings. However, these studies have mostly been carried out independently and the translations between relational and XML models, as well as the impact for probabilistic relational databases of results about query complexity in probabilistic XML and vice versa, have not been made explicit: we detail such translations in this article, in both directions, study their impact in terms of complexity results, and present interesting open issues about the connections between relational and XML probabilistic data models.

Keywords: probabilistic data, relational data, XML.

1 Introduction

A variety of systems have been put forward to represent probabilistic data and cover the needs of the various applications that produce and process uncertain data. In particular, both relational [1] and XML [2] probabilistic data models have been studied in depth, and have been investigated in terms of expressive power, query complexity, underlying algorithms, update capabilities, and so on. Similarly, systems have been developed to query probabilistic relational databases (e.g., MayBMS [3] and Trio [4]) or probabilistic documents (e.g., ProApproX [5] and [6]). By and large, these two lines of work have been conducted independently, and the results obtained have not been connected to each other.

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 121–134, 2013.

The purpose of this article is to give a general overview of probabilistic relational and XML data models, describe the various query languages over these models, present how one can encode queries and probabilistic instances of each one of these models into the other, and investigate the consequences of these encodings in terms of complexity results. We focus specifically on the existence and efficiency of translations across models, and on the transposition of query complexity results, rather than on a systems aspect. Section 2 introduces the probabilistic representation systems we consider, and Section 3 the corresponding query languages. We describe encodings of relations into XML in Section 4, and of XML into relations in Section 5.

2 Data Models

Probabilistic data models are a way to represent a probability distribution over a set of *possible worlds* that correspond to possible states of the data. We focus on the *discrete* and *finite* case where the set of possible worlds is finite; each possible world is associated with a probability value, i.e., a rational in $(0, 1]$, such that the sum of probabilities of all possible worlds is 1. States of the data that are not part of the set of possible worlds have a probability of 0.

A straightforward probabilistic data model is to materialize explicitly the collection of possible worlds with their probability. However, this straightforward representation is not *compact*; it is as large as the number of possible worlds, and any operation on it (such as answering a query) must iterate over all possible worlds. For this reason, probabilistic data models usually represent the set of possible worlds and the probability distribution in an implicit fashion.

Probabilistic data models usually achieve a trade-off between expressiveness and computational complexity: ability to represent as many different kinds of distributions as possible on the one hand, tractability of various operations on the model, such as querying, on the other hand. In this section, we present probabilistic data models for relational data and for XML data: in both settings, we will move from the less expressive to the more expressive.

2.1 Relational Models

Probabilistic models for relational data have usually been built on top of models for representing *incomplete* information. Incomplete information defines a set of possible worlds (the possible completions of the existing information), and probabilistic models usually add some probability scores for each of the possible worlds. See [1] for a general survey of probabilistic relational models.

The tuple-independent model. One of the simplest ideas to define a probabilistic relational model is the *tuple-independent model* [7,8] (also known as tables with *maybe tuples* [9] or *probabilistic ?-tables* [10]). In this model, a probabilistic database is an ordinary database where tuples carry a probability of actually occurring in the database, independently from any other tuple. Formally, given a relational schema Σ, an instance $\widehat{D}$ of the probabilistic relational schema $\widehat{\Sigma}$ is defined as a Σ-instance in which every tuple $R(\mathbf{a})$ is annotated with a rational probability value $\Pr_{\widehat{D}}(R(\mathbf{a})) \in (0,1]$ (with tuples absent from $\widehat{D}$ having probability 0). The probability of a Σ-instance D according to $\widehat{D}$ is then defined as $\Pr_{\widehat{D}}(D) = \prod_{R(\mathbf{a})\in D} \Pr_{\widehat{D}}(R(\mathbf{a})) \prod_{R(\mathbf{a})\notin D} \left(1 - \Pr_{\widehat{D}}(R(\mathbf{a}))\right)$, the product of the probabilities in $\widehat{D}$ of retaining the tuples occurring in D and dropping the others (note that the second product is infinite but has finite support). Since each tuple is either retained or dropped, there are $2^{|\widehat{D}|}$ possible worlds of non-zero probability, and we can check that their probabilities sum to 1.

This model is simple but not very expressive because of the independence assumption. As an example, if the original schema has a predicate $R(A, B)$ with a key constraint $A \to B$, we cannot give non-zero probability to instances $\{R(a, b)\}$ and $\{R(a, b')\}$ without giving non-zero probability to instance $\{R(a, b), R(a, b')\}$ that violates the key constraint.

The block-independent-disjoint model. An extension of the tuple-independent model to support simple mutually exclusive choices is the *block-independent disjoint* model [11,12]. In this variant, we assume that every predicate R of Σ is of the form $R(\mathbf{K}, \mathbf{A})$, where the attributes have been partitioned into two sets: $\mathbf{K}$, the *possible worlds key*, and $\mathbf{A}$, the *value attribute set*. Besides, we require that the key constraint $\mathbf{K} \to \mathbf{A}$ holds. The BID schema $\widehat{\Sigma}$ is defined as before with the added constraint that $\sum_{\mathbf{a}\in\mathbf{A}} \Pr_{\widehat{D}}(R(\mathbf{k}, \mathbf{a})) \leqslant 1$ for every predicate $\widehat{R}(\mathbf{K}, \mathbf{A})$ and possible worlds key $\mathbf{k} \in \mathbf{K}$. Intuitively, for each $\mathbf{k}$, there is a probability distribution on the possible *exclusive* choices of $\mathbf{a}$ (including the default option of choosing no $\mathbf{a}$). A $\widehat{\Sigma}$-instance $\widehat{D}$ defines the probability distribution $\Pr_{\widehat{D}}(D) = \prod_{R(\mathbf{k},\mathbf{a})\in D} \Pr_{\widehat{D}}(R(\mathbf{k}, \mathbf{a})) \prod_{R(\mathbf{k},\bullet)\notin D} \left(1 - \sum_{\widehat{R}(\mathbf{k},\mathbf{a},p)\in\widehat{D}} \Pr_{\widehat{D}}(R(\mathbf{k}, \mathbf{a}))\right)$ with the added constraint that there are no duplicate facts $R(\mathbf{k}, \mathbf{a})$ and $R(\mathbf{k}, \mathbf{a}')$ for any $R(\mathbf{K}, \mathbf{A}) \in \widehat{\Sigma}$, $\mathbf{K} \in \mathbf{K}$, $\mathbf{a}, \mathbf{a}' \in \mathbf{A}$, $\mathbf{a} \neq \mathbf{a}'$ (otherwise the probability is 0).

An example BID database, consisting of a single Customer(**Id**, Name, City) relation (where **Id** is the key) is given in Fig. 1. Mutually exclusive names and cities are given for the two customers, with the possibility also that neither customer exists in a possible world. Such an uncertain table may be obtained, for instance, following a data integration process from conflicting sources.

Intuitively, instances are drawn from a BID instance by picking one of the mutually exclusive choices of $\mathbf{a}$ within each *block* defined by a choice of $\mathbf{k}$, and doing so independently across the blocks. The BID model is more expressive than the tuple-independent model (which can be seen as the case in which all attributes of every predicate are taken as the possible worlds key $\mathbf{K}$).

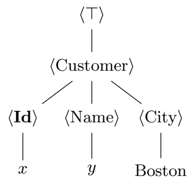
Of course, the structure of the BID model is still unable to represent many kinds of probability distributions. For instance, if instance $\{R(a, b), R(a', b')\}$ has non-zero probability in a relation whose possible worlds key is the first attribute, then instances $\{R(a, b)\}$ and $\{R(a', b')\}$ will also have non-zero probability.

Customer

Id	Name	City	Pr
1	John	New York	0.4
1	Johny	New York	0.2
1	John	Boston	0.1
2	Mary	Boston	0.4
2	Maria	Boston	0.1

Fig. 1. Example BID database

Fig. 2. Example of a TPQJ to encode Customer(x, y, Boston)



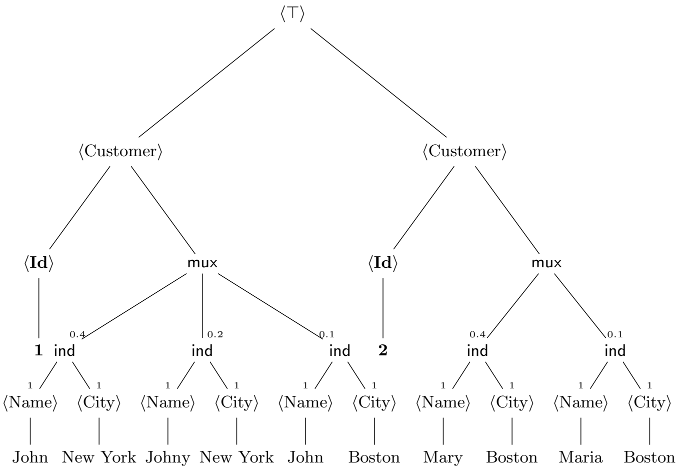


Fig. 3. Example PrXML$^{\{\mathsf{ind},\mathsf{mux}\}}$ tree encoding the BID database of Fig. 1

The pc-tables model. We will now present a much more expressive model: *probabilistic c-tables* (a.k.a. pc-tables) [9,3]. To simplify the writing, we will restrict our exposition to *Boolean pc-tables*, that are known to be as expressive as general pc-tables [9].

Given a relational schema Σ, an instance $\widehat{D}$ of the probabilistic relational schema $\widehat{\Sigma}$ is a Σ-instance which annotates every tuple $R(\mathbf{a})$ with a Boolean formula $\mathrm{Con}_{\widehat{D}}(R(\mathbf{a}))$ over a global finite set of *variables* $\mathcal{V}$ and provides an additional relation B assigning a probability value $\mathrm{Pr}_{\widehat{D}}(x) \in (0, 1]$ to each variable $x \in \mathcal{V}$ occurring in the instance (intuitively, the probability that x is true). Given an assignment ν from the set $\overline{\mathcal{V}}$ of Boolean assignments (i.e., functions from $\mathcal{V}$ to Boolean values $\{\mathfrak{t}, \mathfrak{f}\}$), we define D_ν to be the Σ-instance obtained from $\widehat{D}$ by removing each fact $R(\mathbf{a})$ such that $\mathrm{Con}_{\widehat{D}}(R(\mathbf{a}))$ evaluates to $\mathfrak{f}$ under ν. We then define the probability of assignment ν as: $\mathrm{Pr}_{\widehat{D}}(\nu) = \left(\prod_{x \in \mathcal{V}\,\text{s.t.}\,\nu(x)=\mathfrak{t}} \mathrm{Pr}_{\widehat{D}}(x)\right) \left(\prod_{x \in \mathcal{V}\,\text{s.t.}\,\nu(x)=\mathfrak{f}} \left(1 - \mathrm{Pr}_{\widehat{D}}(x)\right)\right)$. The probability of a Σ-instance D is then: $\mathrm{Pr}_{\widehat{D}}(D) = \sum_{\nu \in \overline{\mathcal{V}}\,\text{s.t.}\,D_\nu = D} \mathrm{Pr}_{\widehat{D}}(\nu)$. Intuitively, Σ-instances are obtained by drawing an assignment of the variables independently according to B and keeping the facts where the condition is true.

It is not very hard to see that *any* probability distribution over *any* finite set of possible worlds can be modeled with a pc-table, given sufficiently many tuples and variables. In particular, BID tables can be expressed as pc-tables, and a direct polynomial-time translation is straightforward.

2.2 XML Models

We will now present probabilistic models for XML data. This presentation is inspired by [2] in which more details can be found. Given an infinite set of labels $\mathcal{L}$, an XML document is a rooted, unordered, directed, and finite tree where each node has a *label* in $\mathcal{L}$. For simplicity, we will always require that the root of an XML document (probabilistic or not) has a fixed label $\sigma_{\mathrm{r}} \in \mathcal{L}$. We denote by $\mathcal{X}$ the collection of XML documents over $\mathcal{L}$.

The $\mathsf{PrXML}^{\{\mathsf{ind}\}}$ *data model.* We define a probabilistic document in the $\mathsf{PrXML}^{\{\mathsf{ind}\}}$ model as an XML document over $\mathcal{L} \cup \{\mathsf{ind}\}$ (ind for *independent*), where the outgoing edges of ind nodes carry a rational number in $(0, 1]$ as label. Such documents will define a probability distribution over $\mathcal{X}$ that we will describe as a sampling process: in a top-down fashion, for every ind node x, independently choose to keep or discard each of its children according to the probability label on each outgoing edge, and, in the father y of x, replace x by the descendants of x that were chosen, removing x and its unchosen descendants. Perform this process independently for every node ind.

Once all ind nodes have been processed and removed in this way, the outcome is ar XML document over $\mathcal{L}$, and its probability is the conjunction of the independent events of all the results of the draws at ind nodes. The probability of a document is the sum of the probability of all outcomes leading to it. Note that multiple different outcomes may lead to the same document.

The $\mathsf{PrXML}^{\{\mathsf{mux}\}}$ *data model.* In the same fashion, we can define the $\mathsf{PrXML}^{\{\mathsf{mux}\}}$ data model (mux for *mutually exclusive*), in which we also require that the outgoing edges of each mux node x carry a rational number in $(0, 1]$ as label such that the labels of all the outgoing edges of x sum to at most 1. The sampling process proceeds as described above, except that each mux node choses at most one of its children according to the probability label on each outgoing edge.

Of course, we can define the $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ data model as the data model in which both ind and mux nodes are allowed, which was studied under the name ProTDB in [13]. An example $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ is given in Fig. 3; as we shall see in Section 4, it is an encoding of the BID database of Fig. 1.

The $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ *data model.* Finally, we define the $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ data model (fie for *formula of independent events*), in which the outgoing edges of fie nodes are labeled with Boolean formulae on some finite set $\mathcal{V}$ of Boolean variables and in which we are given a rational probability value $P(x)$ in $(0, 1]$ for each variable $x \in \mathcal{V}$. The sampling process is to draw independently the truth value of each $x \in \mathcal{V}$ according to the probability $P(x)$, and replace each fie node by its children for which the Boolean formula appearing as an edge label evaluates to t under the assignment that was drawn.

Expressive power and compactness. $\mathsf{PrXML}^{\{\mathsf{ind}\}}$ and $\mathsf{PrXML}^{\{\mathsf{mux}\}}$ are incomparable in terms of expressive power [14]: some probability distributions can be expressed by one and not by the other. Thus, $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ is strictly more expressive than these two, and it is easy to see that one can actually use it to represent *any* finite probability distribution over $\mathcal{X}$ (recall that the root label of all possible documents is fixed).

Any $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ probabilistic document can be transformed in polynomial time into an $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ document (where all Boolean formulae are conjunctions) which yields the same probability distribution: ind nodes can be encoded with fie nodes by creating one Boolean variable for each of their descendants; a similar encoding exists for mux nodes by first encoding n-ary mux nodes in a subtree of binary mux nodes, and then replacing each binary mux node by one fie node whose children are labeled by x and $\neg x$, where x is a fresh Boolean variable with adequate probability.

On the other hand, no polynomial time translation exists in the other direction [14], even when considering $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ probabilistic documents with conjunctions only. In other words, though $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ and $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ have the same expressive power, the latter can be exponentially more compact than the former.

3 Querying Probabilistic Data

Up to this point, we have described probabilistic data models that serve as a concise representation of a probability distribution on a set of possible worlds. However, the main interest of such models is to use them to evaluate queries on all possible worlds simultaneously and return the aggregate results as a probability distribution on the possible answers. For clarity of the exposition, we restrict ourselves to Boolean queries, that are either true or false on a given possible world. Extensions to non-Boolean queries are straightforward.

More formally, given some probabilistic data model, given a probabilistic instance $\widehat{M}$ defining a probability distribution over possible worlds $\mathbf{X}$, and given a Boolean query q that can be evaluated on each $X \in \mathbf{X}$ to produce either $\mathfrak{t}$ or $\mathfrak{f}$, we define the *probability of q on $\widehat{M}$* as: $\hat{q}(\widehat{M}) = \sum_{X \in \mathbf{X} \text{ s.t. } q(X)=\mathfrak{t}} \Pr_{\widehat{M}}(X)$. Evaluating $\hat{q}$ on $\widehat{M}$ means computing the probability of q on $\widehat{M}$. This is called the *possible-worlds query semantics.*

In this section, we present known results about the complexity of query evaluation under this semantics. The query is always assumed to be fixed, i.e., we discuss the *data complexity* of query evaluation.

We will need some basic notions about complexity classes for computation and counting problems [15]. The class FP is that of computation problems solvable in deterministic polynomial time, while $\#\mathsf{P}$ problems are those that can be expressed as the number of accepting runs of a polynomial-time nondeterministic Turing machine. A computation problem is in $\mathsf{FP}^{\#\mathsf{P}}$ if it can be solved in deterministic polynomial time with access to a $\#\mathsf{P}$ oracle. A problem is $\mathsf{FP}^{\#\mathsf{P}}$-hard if there is a polynomial-time Turing reduction from any $\mathsf{FP}^{\#\mathsf{P}}$ problem to it.

3.1 Relational Models

Typical query languages on relational data include *conjunctive queries* (CQs, i.e., select-project-joins), *unions of conjunctive queries* (disjunctions of conjunctive queries, a.k.a. UCQs), and the *relational calculus.* A CQ is *hierarchical* if for any two variables x and y, either the intersection of the set of atoms that contain x with that of y is empty, or one of them is contained in the other (this notion can be extended to arbitrary relational calculus queries, see [1]). A CQ is *without self-join* if all atoms bear distinct relation names.

For instance, a simple CQ for the database of Fig. 1, testing whether any customer is located in Boston, is $q_{\text{Boston}} = \exists x \exists y \, \text{Customer}(x, y, \text{Boston})$. This query, having a single atom, is trivially hierarchical and without self-joins. It is easy to see that $\hat{q}_{\text{Boston}}(\widehat{D}) = 0.55$, where $\widehat{D}$ is the database of Fig. 1.

Extensive results exist about the complexity of evaluating the probability of a query for these various query languages over the tuple-independent, BID, and pc-tables models. We refer to [1] for a detailed overview and summarize here the main results:

- Query evaluation for relational calculus queries over pc-tables is $\mathsf{FP}^{\#\mathsf{P}}$ [16].
- CQs of only one atom are already $\mathsf{FP}^{\#\mathsf{P}}$-hard over pc-tables, even when the Boolean formulae in the pc-table are restricted to conjunctions [17].
- For UCQs over tuple-independent databases, there is a dichotomy between $\mathsf{FP}^{\#\mathsf{P}}$-hard queries and FP queries [18]; however, the only known algorithm to determine the complexity of a query is doubly exponential in the query size, and the exact complexity of this problem is unknown.
- A similar dichotomy, but with a polynomial-time test, holds for CQs without self-joins over BIDs. [16]
- A CQ without self-joins is $\mathsf{FP}^{\#\mathsf{P}}$-hard over tuple-independent databases if and only if it is not hierarchical. [8] Being non-hierarchical is a sufficient condition for any relational calculus query to be $\mathsf{FP}^{\#\mathsf{P}}$-hard.

3.2 XML Models

Tree-pattern queries with joins. The first query language that we will consider on XML data are *tree-pattern queries with joins* (TPQJs). A TPQJ q is a rooted, unordered, directed and finite tree whose nodes are labeled either with elements of $\mathcal{L}$ or with *variable symbols* taken from some infinite fixed set of variables $\mathcal{V}$, and whose edges are either *child* or *descendant* edges. Given an *assignment* $\nu : \mathcal{V} \to \mathcal{L}$, we define the *application* of ν to q (written $q[\nu]$) as the tree where each label $x \in \mathcal{V}$ is replaced by $\nu(x)$. A *match* of q in an XML document d is an assignment ν and a mapping μ from the nodes of $q[\nu]$ to the nodes of d such that:

1. For any child edge $x \to y$ in $q[\nu]$, $\mu(y)$ is a child of $\mu(x)$ in d.
2. For any descendant edge $x \to y$ in $q[\nu]$, $\mu(y)$ is a descendant of $\mu(x)$ in d.
3. For any node x of $q[\nu]$, its label is the same as that of $\mu(x)$ in d.

Intuitively, we match the query with some part of the document so that child, descendant, and fixed label constraints are respected, and all the occurrences of a variable are mapped to nodes with the same label. Tree-pattern queries (TPQs) are TPQJs where all variable symbols are distinct.

We show in Fig. 2 an example TPQJ. Here, x and y are variables, all other labels are from $\mathcal{L}$. We can show that this query, when evaluated on the $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ tree of Fig. 3, yields a probability of 0.55.

Monadic second-order logic with joins. A more expressive query language for XML documents is *monadic second-order tree logic with value joins* (MSOJ). Remember that a monadic second-order formula is a first-order formula extended with existential and universal quantification over node sets. An MSOJ query is a monadic second-order formula over the predicates $x \to y$ (y is a child of x), $\lambda(x)$ (x has label λ), and $x \sim y$ (x and y have same label, a.k.a. *value join*).

A MSO query is an MSOJ query without any value join predicate.

Query complexity. We refer to [2] for a survey of query complexity in probabilistic XML and summarize here the main results:

- Query evaluation for TPQJs over $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ is $\mathsf{FP}^{\#\mathsf{P}}$ [19].
- For MSO queries over $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$, query evaluation is linear-time [20].
- Any non-trivial TPQ is $\mathsf{FP}^{\#\mathsf{P}}$-hard over $\mathsf{PrXML}^{\{\mathsf{fie}\}}$, even when the Boolean formulae of the document are restricted to conjunctions [21].
- If a TPQJ has a single join (i.e., a single variable appears twice in the query), then the following dichotomy holds over $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ [19]: if it is equivalent to a join-free query, it is linear-time; otherwise it is $\mathsf{FP}^{\#\mathsf{P}}$-hard. Testing for this equivalence is Σ_2^{P}-complete. It is open whether this still holds for TPQJs with more than one join.

Note that this list of results has some similarity with that in the relational setting: a broad $\mathsf{FP}^{\#\mathsf{P}}$ membership result, hardness of models with Boolean formulae, even with just conjunctions, and a dichotomy between $\mathsf{FP}^{\#\mathsf{P}}$-hard queries and FP queries for some class of queries over "local" models. In the following, we establish connections between relational and XML settings, exploring whether this yields any connections between these complexity results.

4 From Relations to XML

We explain here how to encode probabilistic relational data models into probabilistic XML.

Encoding instances. Given a relational schema $\Sigma = \{(R_i(A^i_j))\}$, we will define the node labels $\langle \top \rangle$, $\langle R_i \rangle$, $\langle A^i_j \rangle$, along with labels representing all possible constants as text values. The root label of XML documents will always be $\langle \top \rangle$. XML representations of instances of the schema will obey the following DTD:

$$\begin{aligned} \langle \top \rangle &: (\langle R_1 \rangle^*, ..., \langle R_n \rangle^*) \\ \forall i,\ \langle R_i \rangle &: (\langle A^i_1 \rangle, ..., \langle A^i_{n_i} \rangle) \\ \forall i, j,\ \langle A^i_j \rangle &: \#\text{PCDATA} \end{aligned}$$

We now define the encoding $\langle D \rangle$ of an instance D of Σ. The encoding $\langle R_i(a_1, ..., a_{n_i}) \rangle$ of the fact $R_i(a_1, ..., a_{n_i})$ is the subtree whose root has label $\langle R_i \rangle$ and children $\langle A^i_j \rangle$, each child $\langle A^i_j \rangle$ having as child one text node with a label $\langle a_j \rangle$ representing the corresponding a_j. The encoding $\langle D \rangle$ of a full instance D is the XML document whose root has one child $\langle R_i(a_1, ..., a_{n_i}) \rangle$ per fact $R_i(a_1, ..., a_{n_i}) \in D$.

Encoding probabilistic instances. We will now define the encoding $\langle \widehat{D} \rangle$ of probabilistic instances $\widehat{D}$ for the various probabilistic relational data models that we described in Section 2.1. The encodings will be given in the probabilistic XML data models of Section 2.2. When we say that we "encode" one probabilistic data model into another, we mean that the following property holds:

$$\forall \widehat{D}, D, \ \mathrm{Pr}_{\langle \widehat{D} \rangle}(\langle D \rangle) = \mathrm{Pr}_{\widehat{D}}(D) \tag{1}$$

Proposition 1. *Any tuple-independent database can be encoded in linear time as a* $\mathsf{PrXML}^{\{\mathsf{ind}\}}$ *probabilistic document.*

Proof. Given a tuple-independent probabilistic instance $\widehat{D}$, we encode it as a $\mathsf{PrXML}^{\{\mathsf{ind}\}}$ document $\langle \widehat{D} \rangle$ whose root has an ind node as its only child. We root as children of this node the subtrees encoding each of the tuples of $\widehat{D}$, where the probability of the tuple is indicated on the edge attaching it to the ind node. It is easy to see that Equation (1) holds with this encoding. □

Proposition 2. *Any BID can be encoded in linear time as a* $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ *probabilistic document.*

Proof. Consider a BID probabilistic $\widehat{\Sigma}$-instance $\widehat{D}$. We will first define an encoding for the blocks of $\widehat{D}$, before defining the encoding $\langle \widehat{D} \rangle$ of $\widehat{D}$.

For all $R(\mathbf{K}, \mathbf{A}, P) \in \widehat{\Sigma}$, for all $\mathbf{k} \in \mathbf{K}$ such that $R(\mathbf{k}, \mathbf{a}, p) \in \widehat{D}$ for some $(\mathbf{a}, p) \in \mathbf{A} \times P$, we first define $\langle R(\mathbf{k}, \_, \_) \rangle$ as the subtree whose root has label $\langle R \rangle$, has $|\mathbf{K}|$ children $\langle K_j \rangle$ whose children are text nodes $\langle k_j \rangle$ representing the associated k_j, and has as $(|\mathbf{K}| + 1)$-th child a mux node; as children of this mux node, we put one ind node per $\mathbf{a} \in \mathbf{A}$ such that $p = \mathrm{Pr}_{\widehat{D}}(R(\mathbf{k}, \mathbf{a}))$ is > 0, with p as edge label. As children of each of these ind nodes, with edge probability 1, we put $|\mathbf{A}|$ nodes $\langle A_j \rangle$ with children text nodes $\langle a_j \rangle$.

Hence, for each $R(\mathbf{K}, \mathbf{A}, P) \in \widehat{\Sigma}$, for each choice of $\mathbf{k} \in \mathbf{K}$, the mux node will select one of the possible choices of $\mathbf{a} \in \mathbf{A}$ based on their probability in $\widehat{D}$, those choices being independent between all of the mux nodes.

As expected, we define $\langle \widehat{D} \rangle$ to be the $\mathsf{PrXML}^{\{\mathsf{mux}\}}$ document whose root has one child $\langle R(\mathbf{k}, \_, \_) \rangle$ per possible choice of $R(\mathbf{K}, \mathbf{A}, P) \in \widehat{\Sigma}$ and $R(\mathbf{k}, \_, \_) \in \widehat{D}$. □

This construction is illustrated in Fig. 3, which is a $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ encoding of the BID database in Fig. 1.

Proposition 3. *Any pc-table can be encoded in linear time as a* $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ *probabilistic document with the Boolean formulae unchanged.*

Proof. Consider a probabilistic $\widehat{\Sigma}$-instance $\widehat{D}$ in the pc-table formalism. We will set $\langle \mathcal{V} \rangle$ (the variable set for the $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ formalism) to be the variable set $\mathcal{V}$ of $\widehat{D}$, and will simply take $\langle P \rangle(\langle x \rangle)$ to be $\mathrm{Pr}_{\widehat{D}}(x)$.

We now define $\langle \widehat{D} \rangle$ as the $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ document whose root has a fie node as its only child. This node has as descendants the subtrees encoding each of the tuples of $\widehat{D}$, where the condition of the tuple is indicated on the edge attaching it to the fie node. □

Encoding queries. We will now show how a query q on a relational instance D is encoded as a query $\langle q\rangle$ on its XML encoding $\langle D\rangle$. Of course, we will ensure that queries commute with encodings, i.e., the following property holds:

$$\forall \widehat{D}, q,\ q\left(\widehat{D}\right) = \langle q\rangle \left(\left\langle \widehat{D} \right\rangle\right) \tag{2}$$

Proposition 4. *A CQ can be encoded as a TPQJ in linear time.*

Proof. The encoding $\langle a\rangle$ of a constant a is its textual representation, and the encoding $\langle x\rangle$ of a variable x of the query is a TPQJ variable. The encoding $\langle F\rangle$ of an atomic formula $F = R(z_1, ..., z_n)$ over the relation $R(A_1, ..., A_n)$ is the subtree whose root has label $\langle R\rangle$ and has n children; each child has label $\langle A_i\rangle$ and has one child $\langle z_i\rangle$. The encoding $\langle q\rangle$ of a CQ q is a tree whose $\langle\top\rangle$-labeled root has one child $\langle F\rangle$ per atomic formula F in q. □

This encoding is shown in Fig. 2, a TPQJ that encodes q_{Boston}. We rediscover with this result the $\mathsf{FP}^{\#\mathsf{P}}$ membership of CQ evaluation over pc-tables given that of TPQJ over $\mathsf{PrXML}^{\{\mathsf{fie}\}}$. We also obtain the $\mathsf{FP}^{\#\mathsf{P}}$-hardness of TPQJ over $\mathsf{PrXML}^{\{\mathsf{ind}\}}$ given that of CQs over tuple-independent databases. We can finally use this result to find individual hard TPQJ queries as those that are encodings of hard CQs over tuple-independent databases (e.g., non-hierarchical CQs).

Proposition 5. *A query in the relational calculus can be encoded as an MSOJ query in linear time; the resulting query does not have any second-order quantifier, i.e., it is actually a first-order query with joins.*

Proof. Let q be a relational calculus query. We denote by V and C respectively the set of variables and constants appearing in q. For each variable x in V, and each occurrence of x in q, we introduce a new variable x_i. We encode subgoals $R(z_1, ..., z_n)$ for the relation $R(A_1, ..., A_n)$ by the following formula: $\exists y\, \langle R\rangle(y) \wedge \bigwedge_j \exists w\, (y \to w \wedge \langle A_j\rangle(w) \wedge w \to \langle z_j\rangle)$ where the encoding $\langle z_i\rangle$ of a constant a is a fresh variable c_a and the encoding of a variable x is the encoding $\langle x_i\rangle$ of the fresh variable for this specific occurrence of x. Let $\langle q\rangle'$ be the MSO formula obtained from q by encoding all of its subgoals in this way. The MSOJ query $\langle q\rangle$ is $\langle q\rangle' \wedge \left(\bigwedge_{x\in V} \exists x \bigwedge_i x \sim x_i\right) \wedge \left(\bigwedge_{c_a} a(c_a)\right)$ where all x_i's and c_a's are existentially quantified at the top level. □

As an immediate consequence of this encoding, if the original first-order query is *read-once* (no query variable is used twice), it is linear-time to evaluate it over BIDs thanks to the linear-time evaluation of MSO over $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$. Read-once queries are of limited interest, however.

5 From XML to Relations

We now show, in the reverse direction, how to encode probabilistic XML instances into probabilistic relational models. This problem has been explored in [6] with two solutions proposed:

Schema-based: adapted inlining Assume we have a DTD for possible XML documents. This method transforms a $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ probabilistic document (but it is easy to adapt the technique to $\mathsf{PrXML}^{\{\mathsf{fie}\}}$) into a pc-table whose schema is derived from the DTD, adapting techniques from [22] for storing XML documents into relations. Queries are translated as in [22], which may result in queries involving a fixpoint operator in the case of recursive DTDs.

Schemaless: adapted XPath accelerator In the absence of a schema, Hollander and van Keulen propose to transform $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ probabilistic documents (again, the same would hold with $\mathsf{PrXML}^{\{\mathsf{fie}\}}$) into pc-tables by using a pre/size/level encoding as in MonetDB/XQuery [23]. Queries are then translated into queries with inequality predicates and arithmetic.

These two translations are used to show that probabilistic XML can be queried on top of a probabilistic relational engine, Trio in [6] or MayBMS in [24]. However, we present here an alternative translation that has the advantage of transforming TPQJs into CQs, without any need for recursion or inequality predicates.

Encoding instances. Let d be an XML document. We construct the relational schema $\Sigma = \{\mathit{Label}(\mathit{id}, \mathit{lab}), \mathit{Child}(\mathit{id}, \mathit{cid}), \mathit{Desc}(\mathit{id}, \mathit{did}), \mathit{Root}(\mathit{id})\}$ and encode each node n by a unique ID $\langle n \rangle$. The encoding $\langle d \rangle$ of d is a relation over Σ defined as follows:

- for every node n of d with label l, we add a fact $\mathit{Label}(\langle n \rangle, \langle l \rangle)$;
- for every edge (n, n') in d, we add a fact $\mathit{Child}(\langle n \rangle, \langle n' \rangle)$;
- for every node n and descendant n' of n in d, we add a fact $\mathit{Desc}(\langle n \rangle, \langle n' \rangle)$;
- we add a single fact $\mathit{Root}(\langle r \rangle)$ for the root r of d.

Note that this construction is quadratic at worst since we may add linearly many *Desc* facts for every node n.

Encoding probabilistic instances. We will now encode a probabilistic XML document $\hat{d}$ from Section 2.2 into a probabilistic relational instance $\langle \hat{d} \rangle$ of Section 2.1. Again, the goal is to have an encoding that satisfies (1).

We start with a negative result that shows that tuple-independent databases or even BIDs are unusable for encoding even simple probabilistic documents:

Proposition 6. *No probabilistic document of* $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ *with more than one possible world can be encoded as a BID (with the instance encoding above).*

Proof. Assume by way of contradiction we have such a document $\hat{d}$ and its encoding $\langle \hat{d} \rangle$. Since $\hat{d}$ has more than one possible world, it contains at least one ind or mux node, say x, and there is a child y of x such that the edge probability label between x and y is $p < 1$. Let z be the lowest ancestor of x that is neither an ind or mux node; if y is itself an ind or mux node, we take y to be its highest descendant that is neither an ind or mux node.

There exists possible worlds d and d' of $\hat{d}$ such that, in $\langle d \rangle$, $\mathit{Child}(\langle z \rangle, \langle y \rangle)$ and $\mathit{Desc}(\langle z \rangle, \langle y \rangle)$ holds, while in $\langle d' \rangle$, neither of these facts hold. But then, since the *Child* and *Desc* tables are fully independent in a BID, there is a possible world of $\langle \hat{d} \rangle$ where $\mathit{Child}(\langle z \rangle, \langle y \rangle)$ holds and $\mathit{Desc}(\langle z \rangle, \langle y \rangle)$ does not, which is absurd, since no encoding of a possible world of $\hat{d}$ verifies this. □

This result may seem to be an artifact of the particular encoding of instances we chose. However, there is something deeper happening here: BIDs are not able to correlate probabilities of tuples, which means they will not be able in particular to represent *hierarchies of* ind *nodes* [14]. More generally, the main issue comes from the fact that BIDs are not a *strong representation system* [10] for the language of (non-Boolean) conjunctive queries: the output of a conjunctive query over BIDs cannot in general be represented as a BID; on the other hand, $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ can represent any discrete probability distribution, and can therefore represent the output of any query over $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$. This mismatch means it is hopeless to come up with an alternative way of encoding instances that would make BIDs sufficient to encode $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$.

On the other hand, pc-tables can encode probabilistic XML documents:

Proposition 7. *Any* $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ *or* $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ *probabilistic document can be encoded as a pc-table in cubic time.*

Proof. We restrict to $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ since $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ can be tractably encoded into $\mathsf{PrXML}^{\{\mathsf{fie}\}}$. We first remove every fie node of the probabilistic document by connecting each non-fie descendant n' of a fie node to its lowest non-fie ancestor n, labeling the edge (n, n') with the conjunction of all formulae appearing as labels on the edges of the original path from n to n'. We then construct a pc-table from this document as if it were a non-probabilistic document, except that to each tuple $\mathit{Child}(\langle n\rangle, \langle n'\rangle)$ we add the Boolean condition that appears as label on (n, n'), and to each tuple $\mathit{Desc}(\langle n\rangle, \langle n'\rangle)$ we add the conjunction of all Boolean conditions that appear as labels on the edges of the path between n and n'. At worst, this results in a cubic construction: for every node, for every descendant of this node, we have a condition that has at most linear size. □

Encoding queries. Again, our goal is an encoding of tree queries that satisfies (2).

Proposition 8. *A TPQJ can be encoded as a CQ in linear time.*

Proof. Let q be a TPQJ. The CQ $\langle q\rangle$ is the conjunction of the following atomic formulae:

- for every node n of q with constant or variable label l, an atom $\mathit{Label}(\langle n\rangle, \langle l\rangle)$;
- for every child edge (n, n'), an atom $\mathit{Child}(\langle n\rangle, \langle n'\rangle)$
- for every descendant edge (n, n'), an atom $\mathit{Desc}(\langle n\rangle, \langle n'\rangle)$. □

This can be used to reestablish the $\mathsf{FP}^{\#\mathsf{P}}$ membership of TPQJ over $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ from the similar result over relations, or, for example, the $\mathsf{FP}^{\#\mathsf{P}}$-hardness of any encoding of a TPQJ with a single join. Note that the encoding of any TPQ with depth greater than 1 will be non-hierarchical but still tractable on the encodings of $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$: we cannot just use the fact that non-hierarchical queries are intractable since we are working with a specific class of databases.

MSOJ queries cannot be encoded into the relational calculus, since they can express such things as the existence of a path between nodes of a graph (this graph being represented as a tree, e.g., as in Section 4), which is impossible to test in first-order logic. [25]

6 Conclusion

We have thus established connections between probabilistic relational and XML models, showing how probabilistic instances and queries from one model can be encoded into the other. Though we can rediscover some general results in this way (such as the $\mathsf{FP}^{\#\mathsf{P}}$-completeness of query evaluation), we also see that results over probabilistic XML (such as the linear-time evaluation of MSO queries, or the dichotomy for TPQJ queries with a single join) are not direct translations of similar results in the relational case but deep consequences of the true structure.

To go further, one direction would be to look at the tree-width (of the data structure, of the queries, of the Boolean formulae) as an indicator of the tractability of a query; Jha and Suciu have shown [26] that it is tractable to evaluate the probability of a bounded tree-width Boolean function. It is still an open problem to understand the dependency between the tree-width of a query lineage and the tree-width of the query, of the data and of the Boolean formulae. This could suggest new tractable classes of probabilistic relational databases, inspired by the tractable classes of probabilistic XML.

We have restricted our study to discrete finite probabilistic distributions; models for discrete infinite distributions arise naturally in probabilistic XML [27] by adding probabilities to an XML schema [28]; their meaning is less clear in the relational setting. Probabilistic models with continuous distributions can also be defined in the relational [29] and XML [30] cases, though the precise semantics can be tricky. Moreover, no strong representation systems (say, for conjunctive queries) involving continuous distributions have been put forward yet.

References

1. Suciu, D., Olteanu, D., Ré, C., Koch, C.: Probabilistic Databases. Morgan & Claypool (2011)
2. Kimelfeld, B., Senellart, P.: Probabilistic XML: Models and complexity (September 2011) Preprint available at, `http://pierre.senellart.com/publications/kimelfeld2012probabilistic`
3. Huang, J., Antova, L., Koch, C., Olteanu, D.: MayBMS: a probabilistic database management system. In: SIGMOD (2009)
4. Widom, J.: Trio: A system for integrated management of data, accuracy, and lineage. In: CIDR (2005)
5. Souihli, A., Senellart, P.: Optimizing approximations of DNF query lineage in probabilistic XML. In: Proc. ICDE (April 2013)
6. Hollander, E., van Keulen, M.: Storing and querying probabilistic XML using a probabilistic relational DBMS. In: MUD (2010)
7. Lakshmanan, L.V.S., Leone, N., Ross, R.B., Subrahmanian, V.S.: ProbView: A flexible probabilistic database system. ACM Transactions on Database Systems 22(3) (1997)
8. Dalvi, N.N., Suciu, D.: Efficient query evaluation on probabilistic databases. VLDB Journal 16(4) (2007)
9. Green, T.J., Tannen, V.: Models for incomplete and probabilistic information. In: Proc. EDBT Workshops, IIDB (March 2006)

10. Sarma, A.D., Benjelloun, O., Halevy, A.Y., Widom, J.: Working models for uncertain data. In: ICDE (2006)
11. Barbará, D., Garcia-Molina, H., Porter, D.: The management of probabilistic data. IEEE Transactions on Knowledge and Data Engineering 4(5) (1992)
12. Ré, C., Suciu, D.: Materialized views in probabilistic databases: for information exchange and query optimization. In: VLDB (2007)
13. Nierman, A., Jagadish, H.V.: ProTDB: Probabilistic data in XML. In: VLDB (2002)
14. Abiteboul, S., Kimelfeld, B., Sagiv, Y., Senellart, P.: On the expressiveness of probabilistic XML models. VLDB Journal 18(5) (October 2009)
15. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
16. Dalvi, N.N., Suciu, D.: Management of probabilistic data: foundations and challenges. In: PODS (2007)
17. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science 8 (1979)
18. Dalvi, N.N., Schnaitter, K., Suciu, D.: Computing query probability with incidence algebras. In: PODS (2010)
19. Kharlamov, E., Nutt, W., Senellart, P.: Value joins are expensive over (probabilistic) XML. In: Proc. LID (March 2011)
20. Cohen, S., Kimelfeld, B., Sagiv, Y.: Running tree automata on probabilistic XML. In: PODS (2009)
21. Kimelfeld, B., Kosharovsky, Y., Sagiv, Y.: Query evaluation over probabilistic XML. VLDB Journal 18(5) (2009)
22. Shanmugasundaram, J., Tufte, K., Zhang, C., He, G., DeWitt, D.J., Naughton, J.F.: Relational databases for querying XML documents: Limitations and opportunities. In: VLDB (1999)
23. Boncz, P.A., Grust, T., van Keulen, M., Manegold, S., Rittinger, J., Teubner, J.: MonetDB/XQuery: a fast XQuery processor powered by a relational engine. In: SIGMOD (2006)
24. Stapersma, P.: Efficient query evaluation on probabilistic XML data. Master's thesis, University of Twente (2012)
25. Libkin, L.: Elements of Finite Model Theory. Springer (2004)
26. Jha, A.K., Suciu, D.: On the tractability of query compilation and bounded treewidth. In: ICDT (2012)
27. Benedikt, M., Kharlamov, E., Olteanu, D., Senellart, P.: Probabilistic XML via Markov chains. Proceedings of the VLDB Endowment 3(1) (September 2010)
28. Abiteboul, S., Amsterdamer, Y., Deutch, D., Milo, T., Senellart, P.: Finding optimal probabilistic generators for XML collections. In: Proc. ICDT (March 2012)
29. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: SIGMOD (2003)
30. Abiteboul, S., Chan, T.H.H., Kharlamov, E., Nutt, W., Senellart, P.: Capturing continuous data and answering aggregate queries in probabilistic XML. ACM Transactions on Database Systems 36(4) (2011)

On Bridging Relational and Document-Centric Data Stores

John Roijackers and George H.L. Fletcher

Eindhoven University of Technology, The Netherlands
john@roijackers.net, g.h.l.fletcher@tue.nl

Abstract. Big Data scenarios often involve massive collections of nested data objects, typically referred to as "documents." The challenges of document management at web scale have stimulated a recent trend towards the development of document-centric "NoSQL" data stores. Many query tasks naturally involve reasoning over data residing across NoSQL and relational "SQL" databases. Having data divided over separate stores currently implies labor-intensive manual work for data consumers. In this paper, we propose a general framework to seamlessly bridge the gap between SQL and NoSQL. In our framework, documents are logically incorporated in the relational store, and querying is performed via a novel NoSQL query pattern extension to the SQL language. These patterns allow the user to describe conditions on the document-centric data, while the rest of the SQL query refers to the corresponding NoSQL data via variable bindings. We give an effective solution for translating the user query to an equivalent pure SQL query, and present optimization strategies for query processing. We have implemented a prototype of our framework using PostgreSQL and MongoDB and have performed an extensive empirical analysis. Our study shows the practical feasibility of our framework, proving the possibility of seamless coordinated query processing over relational and document-centric data stores.

1 Introduction

Nested data sets are ubiquitous in Big Data scenarios, such as business and scientific workflow management [2,12] and web analytics [15]. The massive collections of such loosely structured data "documents" encountered in these scenarios have stimulated the recent emergence of a new breed of document-centric "NoSQL" data stores, specifically targeting the challenges of data management at web-scale [6,17]. These solutions aim for flexible responsive management of nested documents, typically serialized in the JSON [8] data format.

While very popular and successful in many Big Data application domains, NoSQL systems will of course not displace traditional relational stores offering structured data storage and declarative querying (e.g., SQL-based access). Indeed, there is growing consensus that both NoSQL and "SQL" systems will continue to find and fulfill complementary data management needs [17]. Therefore, we can expect it to become increasingly common for users to face practical scenarios where they need to reason in a coordinated fashion across both

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 135–148, 2013.

document-centric and relational stores. Indeed, the study presented in this paper was sparked directly by our work with a local web startup.

To our knowledge, there currently exists no unified query framework to support users in such cross-data-store reasoning. Hence, users must currently issue independent queries to the separate relational and document stores, and then manually process and combine intermediate results. Clearly, this labor-intensive and error-prone process poses a high burden to consumers of data.

Contributions. In this paper, we take the first steps towards bridging the worlds of relational and document data, with the introduction of a novel general framework, and its implementation strategies, for seamless querying over relational and document stores. Our particular contributions are as follows:

- we present a logical representation of NoSQL data in the relational model, and a query language extension for SQL, which permits seamless querying of NoSQL data in SQL;
- we develop a basic approach for processing such extended queries, and present a range of practical optimizations; and,
- we present the results of a thorough empirical analysis of our framework, demonstrating its practicality.

The solution we present here is the first to eliminate the need for ad-hoc manual intervention of the user in query (re)formulation and optimization.

Related Work. Our logical representation of documents is inspired by the so-called first-order normal form for tabular data [11] and its applications in web data integration [1,3,9]. Furthermore, our language extension is motivated by the successes of syntactic extensions for SQL which have been explored, e.g., for RDF graphs [7]. To our knowledge, however, our novel proposals are the first to directly address the new data and query challenges, resp., raised by NoSQL stores. The only closely related effort in this area that we are aware of is the recently proposed SOS project [4]. SOS aims at providing a uniform application programming interface to federations of heterogeneous NoSQL data stores, which is complementary to the goals of our framework.

Organization. We proceed in the paper as follows. In the next section, we introduce our theoretical framework. We then discuss query processing strategies in Sec. 3. In Sec. 4 we present the setup of our empirical study, and then in Sec. 5 we present results of our experiments. Finally, we close the paper with indications for further research in Sec. 6.

2 Theoretical Framework

In this section we present our generic framework for query processing solutions over SQL and NoSQL data stores. There are two major components of the framework. First, a logical relation available in the SQL database representing (arbitrary) NoSQL data is introduced in Sec. 2.1. Second, to seamlessly and transparently query the NoSQL data from SQL, we present an SQL query language extension with a JSON-like syntax in Sec. 2.2. We conclude the section with an overview of the entire theoretical framework.

2.1 Logical Representation of NoSQL Data in the Relational Model

In this paper, we model NoSQL *documents* as finite sets of key-value pairs, where values themselves can be finite sets of key-value pairs, and where keys and atomic values are elements of some infinite universe (e.g., Unicode strings). For the sake of simplicity, and without any loss of generality, we omit other features available in the JSON data model, such as ordered list values.

Disregarding implementation details for the moment, we assume that in the relational store there is a relation $F\,(id, key, value)$ available containing records representing the NoSQL data. This is a potentially virtual relation, i.e., F is implemented as a non-materialized view on the NoSQL data. The basic idea is that each document d in the NoSQL data set is assigned a unique identifier i_d. Then, for each key-value pair $k : v$ of d, a triple (i_d, k, v) is added to F. If v is a nested value, then it is assigned a unique identifier, and the process recurs on the elements of v.

We illustrate F via an example. Suppose we have product data in the document store, with nested supplier data to identify the supplier and to keep track of current stock. Then an example snippet of F, containing three product documents i_1, i_2, and i_3, is as follows:

$$\begin{aligned} F \;=\{&(i_1, \text{name}, \textit{monitor}), (i_1, \text{category}, 7), (i_1, \text{color}, \textit{black}), (i_1, \text{supplier}, i_4), \\ &(i_2, \text{name}, \textit{mouse}), (i_2, \text{category}, 37), (i_2, \text{color}, \textit{pink}), (i_2, \text{supplier}, i_5), \\ &(i_3, \text{name}, \textit{keyboard}), (i_3, \text{category}, 37), (i_3, \text{supplier}, i_6), \\ &(i_4, \text{id}, 1), (i_4, \text{stock}, 1), (i_5, \text{id}, 3), (i_5, \text{stock}, 5), (i_6, \text{id}, 1)\}. \end{aligned}$$

For example, i_1 corresponds to the document

$$\{\text{name} : \textit{monitor}, \text{category} : 7, \text{color} : \textit{black}, \text{supplier} : \{\text{id} : 1, \text{stock} : 1\}\}.$$

Here, we see that the "supplier" key of i_1 has a nested value, given the unique identifier i_4. Clearly, the F representation of a document store is well-defined. Due to space limitations we omit a formal definition and refer the reader to the full version of this paper for further details [16].

The F representation has two important advantages. Firstly, it is a basic and extremely flexible way to describe data which can be used to denote any possible type of NoSQL data, including documents with missing and/or set-valued keys. Secondly, F has a fixed schema and thus the (potentially schema-less) non-relational data can be accessed via a standard table in the SQL database.

The relation F can be used in queries just like any other relation in the relational database. This means we can join F to other relations to combine SQL and NoSQL data to construct a single query result. Important to note however, is that the records in F are retrieved from the external NoSQL data source on the fly. To return the records of F, the SQL database has to retrieve the external NoSQL data in triple format.

This implies that the retrieved data has to be communicated to the relational database, which is dependent on the exact implementation of F and the method used to retrieve data from the NoSQL source. We assume that information regarding to which NoSQL source F should connect and what data to retrieve is given. In practice, the implementation of F will of course be parameterized, to specify which NoSQL database should be used. For readability however, we ignore such parameters. In the remainder of this paper we will thus simply use F as the relation of triples that represents the NoSQL data. We discuss implementation details further in Sec. 4, below.

2.2 Declarative Querying over SQL and NoSQL Stores

A series of self joins of F on the *id* field allows reconstruction of the NoSQL data. However, in cases where many self joins are needed (e.g., to retrieve data elements in deeply nested documents) this is a tedious error-prone task. To facilitate querying the NoSQL data more conveniently and avoid the task of manually adding join conditions, we introduce an SQL extension which we call a *NoSQL query pattern* (NQP).

An NQP is based on the concept of variable bindings as used in standard conjunctive query patterns which are at the core of well-known languages such as Datalog, SQL, SPARQL, and XPath [1]. Consider a graph pattern where triples can be nested, and triple elements can be either variables or atomic values. All triples on the same nesting level describe the same triple subject and therefore the identifier of this element can be excluded from the query. The result is a JSON-like nested set of key-value pairs where some elements may be variables.

For example, suppose that on the SQL side we have a table of product suppliers, with schema *Supplier*(*id*, *name*, *region*), and on the NoSQL side the previously mentioned product data (this is an actual real-world situation which we encountered in our collaboration with a local web startup). The query given in Listing 1.1 retrieves product name and supplier region information for products in category 37 having a minimum stock of 2.

```
SELECT
  p.n, s.region
FROM
  NoSQL( name: ?n, category: 37, color: ?c,
         supplier: ( id: ?i, stock: ?s ) ) AS p,
  Supplier AS s
WHERE
  p.i = s.id AND p.s >= 2
```

Listing 1.1. Example SQL query with an NQP

Besides illustrating the NQP syntax, the example also demonstrates how NQP's are included in an SQL query. This method allows easy isolation of the

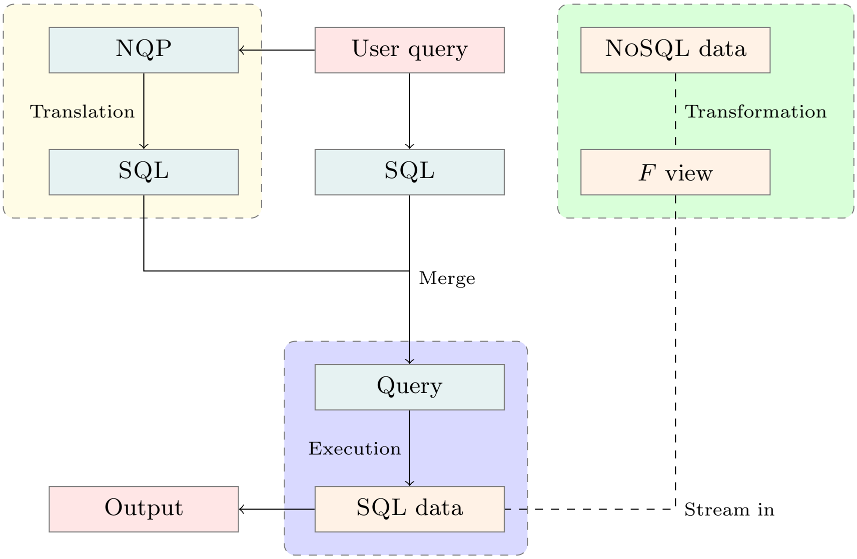


Fig. 1. Architectural workflow illustrating the life of a SQL+NoSQL query

NQP and references to the variable bindings in the rest of the query. In the SQL part of the query the variables in the NQP can be treated as relation attributes from the virtual NoSQL relation. In Sec. 3 we describe a method to automatically translate an arbitrary NQP to a pure SQL equivalent.

The NQP extension provides a convenient way to describe which NoSQL data is required and how it should be reconstructed on the SQL side, without exposing to the user the underlying representation and query processing. As far as the user is concerned, the NoSQL data is contained in a single native relation.

In addition to user transparency and simplicity, an additional major advantage of our approach is its independence from the underlying NoSQL database. The use of another document store implies a new implementation of the F relation. However, SQL+NQP queries, and hence the applications in which they are embedded, are not affected by the change of the underlying database.

Architectural Overview. To summarize, we give an overview of our theoretical framework in Fig. 1. Documents, stored in a NoSQL database, are made available in the relational database via a logical relation F. This data is streamed in on demand, so the NoSQL database is still used as the primary store for the document data. A user can then issue an SQL query containing NQP's, which describes constraints on the desired NoSQL data and their combination with the SQL data. The query is then automatically translated behind the scenes to a pure SQL query and processed by the relational database.

3 Query Processing Strategies

In this section we introduce implementation strategies for putting our theoretical framework of the previous section into practice. The NQP extension, based on a virtual relational representation of document data, must be automatically translated to a pure SQL equivalent to be executed by an SQL database. In Sec. 3.1 we provide a baseline approach to this translation. Sec. 3.2 focuses on optimization techniques to accelerate the performance of this base translation.

3.1 Base Translation

The nested key-value pairs in an NQP can be recursively numbered as follows:

$$\text{NoSQL}(k_1 : v_1, k_2 : v_2, \ldots, k_r : (k_{r,1} : v_{r,1}, k_{r,2} : v_{r,2}, \ldots, k_{r,m} : v_{r,m}) \ldots, k_n : v_n)$$

The basic idea behind a translation of an NQP is that each key-value pair $t = k_t : v_t$ is represented in SQL using its own copy F_t of the logical relation F. For these copies we introduce a shorthand notation to avoid naming conflicts and to apply selection conditions, specified as follows:

F_t	**Constant** k_t	**Variable** k_t
Constant v_t	$\rho_{F_t(\text{i}_t,\text{k}_t,\text{v}_t)}(\sigma_{\text{key}=k_t \wedge \text{value}=v_t}(F))$	$\rho_{F_t(\text{i}_t,\text{k}_t,\text{v}_t)}(\sigma_{\text{value}=v_t}(F))$
Variable v_t	$\rho_{F_t(\text{i}_t,\text{k}_t,\text{v}_t)}(\sigma_{\text{key}=k_t}(F))$	$\rho_{F_t(\text{i}_t,\text{k}_t,\text{v}_t)}(F)$
Nested v_t	$\rho_{F_t(\text{i}_t,\text{k}_t,\text{v}_t)}(\sigma_{\text{key}=k_t}(F))$	$\rho_{F_t(\text{i}_t,\text{k}_t,\text{v}_t)}(F)$

Each copy of F is subscripted with t, just like each of its attributes. Depending on the type of both the key and value of the pair, we can add a selection operator to exclude unnecessary triples from F_t on the SQL side. When neither the key nor the value is a constant value, F_t is only a renamed version of F. This is inefficient for query processing, because F_t then contains all triples in F, and thus potentially the entire NoSQL dataset.

The next translation step is reconstruction of the NoSQL data by correctly joining the F_t copies. For key-value pairs at the same nesting level in the NQP this means that the *id* attributes should have the same value, since the F_t relations for an NoSQL object have been given identical *id* values. For a nested NQP the translation is applied recursively. The triple relations are created such that nested data is connected via a *value* attribute that is equal to the *id* value of the nested triples. Combining the correct triples is therefore only a matter of inserting the correct join condition for the nested set of F copies.

Using this method, the NQP of the example query from Listing 1.1 is translated to the following series of joins:

$$F_{\text{name}} \bowtie_{\text{i}_{\text{name}}=\text{i}_{\text{category}}} F_{\text{category}} \bowtie_{\text{i}_{\text{category}}=\text{i}_{\text{supplier}}} F_{\text{supplier}} \bowtie_{\text{v}_{\text{supplier}}=\text{i}_{\text{supplier.id}}}$$
$$\big(F_{\text{supplier.id}} \bowtie_{\text{i}_{\text{supplier.id}}=\text{i}_{\text{supplier.stock}}} F_{\text{supplier.stock}}\big) \bowtie_{\text{i}_{\text{supplier}}=\text{i}_{\text{color}}} F_{\text{color}}.$$

Note that since the nested relations $F_{r,t}$ have equal *id* values, a single join constraint is sufficient. Because of the introduced shorthand notation, this query applies the correct selection criteria to each relation F_t, renames the attributes and combines the corresponding triples.

In the rest of the query outside of the NQP, references to variables in the NQP occur. This means that we must keep track of the variables used in the NQP and use this information to adjust the SQL part of the query accordingly. To achieve this we introduce the function *insert*(v, a) defined as:

$$\mathit{insert}(v, a) = \begin{cases} V_v = \{a\}\,, & \text{if } V_v \text{ does not exist} \\ V_v = V_v \cup \{a\}\,, & \text{otherwise.} \end{cases}$$

We call this function for each variable in the NQP. Now, letting $\mathcal{V}$ be the set of variables encountered, we can replace the references to these variables in the SQL part of the query by an attribute from a triple relation copy as follows: $\forall_{v \in \mathcal{V}} Q\left[\text{r}.v := e_v\right]$, for an arbitrary $e_v \in V_v$.

Now the NQP has been translated, all relations are joined correctly, and variable references in the SQL part of the query have been replaced by appropriate relation attributes. Finally, we add an additional selection condition to the query which ensures that if the same variable is used multiple times in the NQP, it has an equal value. Formally and without minimizing the number of equalities this means: $\bigwedge_{v \in \mathcal{V}} \bigwedge_{i,j \in V_v} i = j$.

3.2 Optimizations

We next briefly introduce optimization strategies, to improve upon the naive base translation of the previous section. Further details can be found in [16].

(1) Data filtering. Although the naive approach from Sec. 3.1 results in a correct SQL translation, this method has a significant disadvantage. For each copy of F_t the entire NoSQL dataset must be transformed to triples and shipped to the SQL database. A straightforward way to improve performance is to filter NoSQL data that is not required to answer the query.

Firstly, we introduce a parameter c for the relations F_t. This parameter is used to describe a conjunction of selection conditions. These conditions are pushed down to the NoSQL database to restrict the number of documents that have to be sent to the SQL database. Also, in addition to conventional relational algebra conditions, more advanced selection conditions, like the existence of a field in the NoSQL database, can be pushed down:

$$c = \begin{cases} k_t = v_t, & \text{if both } k_t \text{ and } v_t \text{ have a constant value} \\ \mathit{exists}(k_t), & \text{if only } k_t \text{ has a constant value} \\ \mathit{true}, & \text{otherwise.} \end{cases}$$

Moreover, while translating the NQP to SQL, we can collect all selections and combine them in a single condition that describes the entire constraint on the NoSQL data we can derive from the NQP. Because this is a conjunction of criteria, more documents are filtered from each F_t copy.

Additionally, we derive new NoSQL selection conditions from the SQL part of the query. Moreover, we can also apply transitivity on the available conditions to put even more selective constraints on the NoSQL data.

For our example this means that the SQL selection `p.s >= 2` can be included in the final selection condition used to filter the NoSQL documents: $c = \textit{exists}(\text{name}) \wedge \text{category} = 37 \wedge \textit{exists}(\text{color}) \wedge \textit{exists}(\text{supplier}) \wedge \textit{exists}(\text{supplier.id}) \wedge \text{supplier.stock} \geq 2$. Note that we adopt a non-null semantics for simplicity.

This *document filter pushdown* reduces the number of NoSQL documents. The matching NoSQL objects themselves, however, are completely transformed to triples before they are shipped to the SQL side. We can further reduce the size of the NoSQL data by excluding triples representing attributes that are not used elsewhere in the SQL query. Hence, we introduce a parameter p for the relations F_t, to describe an *attribute filter pushdown* condition on the NoSQL data to eliminate shipping of these unused attributes.

To determine if a given key-value pair t must be shipped to the SQL database, we recursively determine if t contains a variable used in the SQL part of the query, or, in case v_t is nested, if there exists a nested key-value pair under t which should be in the translation. Note that the F_t copy representing t is no longer required in the query. In addition to decreasing the amount of data that has to be shipped from NoSQL to SQL, this optimization will lead to a final query which also contains fewer relations and thus requires fewer joins to reconstruct the NoSQL data in the relational database.

For the example query this means we can exclude F_{color} and F_{category}, since these are not used except as selection conditions on the NoSQL data. The reduction of the number of triples is achieved by the following projection argument: $p = \{\text{name}, \text{supplier}, \text{supplier.id}, \text{supplier.stock}\}$. Note that F_{supplier} is still necessary to retrieve the supplier and its current stock count.

(2) Temporary table. Each relation copy F_t requires a new connection to the external NoSQL database. To avoid this overhead we can create a temporary relation T equal to F prior to a query execution. We then change the translation such that instead of F the temporary relation T is used. As a result all triples are communicated to the SQL database only once. Document and attribute filter pushdown can now be applied to this temporary table.

(3) Tuple reconstruction. Triples are joined based on shared *id* values, matching a higher level *value* in case of nested data. This means that for nested data it is not possible for an arbitrary pair of triples in F to determine whether or not they belong to the same NoSQL data object. This information could be used when joining the triples and thereby speed up the NoSQL data reconstruction in the relational database.

To achieve this, we add an additional attribute *nosql* to F that indicates the NoSQL document to which the record belongs. Triples originating from the same document get equal *nosql* values, using the native NoSQL document identifier. Only triples from the same NoSQL document are combined on the SQL side, so this extra information can speed up the join.

At the creation of the temporary table we use the *nosql* attribute to include additional selection conditions. We create a hash index on the *nosql* attribute and include an equality selection condition on this attribute for each F_t used in the SQL query. This way, the relational database has additional knowledge about the triple structure and is able to combine triples that belong to the same NoSQL document, regardless of the nesting level.

4 Experimental Setup

With the developed theoretical framework and the automatic query translation and its optimizations, we implemented a prototype version of the hybrid database to provide a proof of concept that the theoretical framework is practically feasible. In this section, we first elaborate on the specific SQL and NoSQL databases we used and how the virtual relation F is implemented. Following this, we then discuss the experiment setup, including the data and benchmark queries used in our study. Interested readers will find deeper details on all aspects of the experimental setup in the full version of this paper [16].

Environment. In our empirical study we use MongoDB 2.0.2, an open-source and widely used document-oriented NoSQL data store.[1] For the relational database we use PostgreSQL 9.1.2.[2], an open source, industrial strength, widely used system, which offers great extensibility. The latter is particularly useful for the implementation of F.

The logical F relation is implemented as a foreign table, using a foreign data wrapper (FDW) as described in the SQL/MED extension of the SQL standard [14]. From a user perspective a foreign table is a read-only relation similar to any other relation. The technical difference is that the data in a foreign table is retrieved from an external source at query time. We use the Multicorn 0.0.9 FDW to implement the foreign table.[3] This is an existing PostgreSQL extension that uses the output of a Python script as the external data source. In our case the script retrieves the data from MongoDB and performs necessary transformations on the documents.

For the experiment we use a machine with 2 quad core Intel Xeon E5640 processors, 36 GB of memory, and 4 10 krpm SAS hard disks in RAID 5 running on Debian wheezy/sid for the PostgreSQL and MongoDB database. Due to practical constraints a virtual machine on the same machine is responsible for running the experiment. The virtual machine runs on the same OS, uses 2 processor cores and has 2 GB of memory available. The experiment process sends a query to be executed to the main machine, retrieves the result, and performs time measurements.

[1] http://www.mongodb.org/

[2] http://www.postgresql.org/

[3] http://multicorn.org/

Experiments. The experiment is twofold. We start with a small *Experiment 1* in which we focus on the feasibility of the prototype implementation and the effect of using a temporary relation prior to the actual query execution. We compare implementations $\mathcal{I}_a$ and $\mathcal{I}_b$. Implementation $\mathcal{I}_a$ is the base translation with only the data filtering optimization included. In $\mathcal{I}_b$ the temporary table is also implemented. For both implementations the number of documents returned is limited to 100.

In *Experiment 2* we compare two implementations to see what the impact of the tuple reconstruction optimization is in a larger scale setting. Here we have implementation $\mathcal{I}_c$, similar to $\mathcal{I}_b$, and implementation $\mathcal{I}_d$ with the tuple reconstruction included. For this experiment we increase the limit on the number of MongoDB documents returned to 25 000.

For both experiments the performance of the implementations is measured in terms of query execution time. Query translation, logical table generation, indexing, and actual query execution are measured separately.

Data. We use multiple datasets for a detailed comparison. The first dataset is constructed using a set of real life product data taken from a company's production environment. This NoSQL data is similar to our running example above, and can be joined to two relations in the SQL database.

Using this product set as a basis, we create different datasets by varying three parameters. Firstly, the number of products in the NoSQL dataset is either low or high. In our case this means 100 000 or 400 000 documents respectively. Also, we vary the SQL data size between 1000 and 10 000 records. And finally we cover different join probability between SQL and NoSQL data by using 0.05, 0.20, and 1.00. We use all possible combinations of these variations and thus create 12 datasets based on the product data. Each of these datasets is named $\mathcal{S}_{n,s,j}$, where n, s, and j describe the NoSQL size, SQL size, and join probability respectively.

To see the impact of a totally different dataset, we also use a *Twitter* dataset. For this dataset we collected a coherent set of 500 000 tweets about a single subject posted between March 5 and March 11 of 2012 using the *Twitter Search API*. These tweets have information about users and languages included, which we store in an SQL database to create a possibility to join the non-relational *Twitter* data to SQL data. We use $\mathcal{S}_t$ to denote this dataset.

Queries. The manner in which a query uses SQL and NoSQL data influences the query execution time. There are different ways to combine data from both stores regarding the data source on which the query mainly bases its selections. We use the term *flow class* to indicate how the query is constructed. We consider four flow classes: $\mathcal{F}_{\text{i}}$ (queries only selecting NoSQL data); $\mathcal{F}_{\text{ii}}$ (queries selecting SQL data on the basis of NoSQL data); $\mathcal{F}_{\text{iii}}$ (queries selecting NoSQL data on the basis of SQL data); and, $\mathcal{F}_{\text{iv}}$ (queries selecting SQL data on the basis of NoSQL data selected using SQL data). Other flow classes can be studied, but these cover the basic ways to combine data.

Besides the flow class of a query, query execution can also depend on how the NoSQL data is used in the SQL query. We distinguish two query properties: (a) uses all mentioned NoSQL keys in the SQL part of the query; and, (b) all mentioned NoSQL keys exist in the data for all documents. We create different *query types* based on the possible combinations of these two properties, namely: $\mathcal{Q}_1$ $((a) \wedge (b))$; $\mathcal{Q}_2$ $((a) \wedge \neg(b))$; $\mathcal{Q}_3$ $(\neg(a) \wedge (b))$; and, $\mathcal{Q}_4$ $(\neg(a) \wedge \neg(b))$. The example query of Listing 1.1 above falls into flow class $\mathcal{F}_{\mathrm{ii}}$ and query type $\mathcal{Q}_4$.

For each dataset we construct a set of query templates, for each combination of a flow class and a query type. For each dataset we thus have a total of 16 query templates. From each query template 16 concrete queries are created by filling in random values for placeholders included in the template. In our experiments, we evaluate and measure each concrete query 5 times. For the empirical analysis we drop the highest and lowest value from the 5 repetitions and report the average; we also drop the 3 highest and lowest averages per template.

5 Empirical Analysis

In this section we discuss the results of Experiments 1 and 2. We start with a presentation of the results of the first experiment, where we analyze the effect of using a temporary relation. This is followed by a discussion of the second experiment, where the NoSQL data reconstruction optimization is used. Finally, we discuss the main conclusions which can be drawn from these investigations.

Results of Experiment 1. In Experiment 1 we look at the effect of creating a single temporary relation containing all required triples prior to the actual query execution on the SQL side in a small scale experiment. In Table 1 the average result per query flow class and query type are presented. From these results, it is clear that the use of a temporary table is a significant improvement, except for $\mathcal{Q}_3$ within $\mathcal{F}_{\mathrm{i}}$. For this exception however, the initial result for $\mathcal{I}_a$ was already low and the result for $\mathcal{I}_b$ still is the best for all flow class and query type combinations.

In Table 2 we see that the *Twitter* dataset in particular profits from this optimization. For *Twitter*, the average query time was 21.7521 s compared to 0.4785 s with the use of a temporary relation. For the average product dataset, on the other hand, the average result dropped from 9.8303 s to 0.5789 s.

Results of Experiment 2. The larger-scale second experiment focuses on the effects of optimizing the NoSQL data reconstruction in the relational database by including additional information with each triple. Similar to the analysis of Experiment 1, Table 3 provides an overview per query flow class and query type.

For $\mathcal{F}_{\mathrm{i}}$ and $\mathcal{F}_{\mathrm{ii}}$ the added SQL attribute creates a small overhead. As a result, the tuple reconstruction is not an improvement and hence we omit their details in the table. For $\mathcal{F}_{\mathrm{i}}$ and $\mathcal{F}_{\mathrm{ii}}$ we observed that the average time increased from 11.5800 s and 12.5575 s to 12.4004 s and 13.1219 s respectively.

The other flow classes, $\mathcal{F}_{\mathrm{iii}}$ and $\mathcal{F}_{\mathrm{iv}}$, indicate that optimizing the tuple reconstruction can have a positive effect on the performance of the prototype. Both

Table 1. Comparison of $\mathcal{I}_a$ and $\mathcal{I}_b$ performance in seconds per flow class

(a) Flow class $\mathcal{F}_{\text{i}}$

	$\mathcal{I}_a$	$\mathcal{I}_b$	$\mathcal{I}_b$ / $\mathcal{I}_a$
$\mathcal{Q}_1$	4.1123	0.4766	0.116
$\mathcal{Q}_2$	22.0757	0.6694	0.030
$\mathcal{Q}_3$	0.2467	0.4021	1.630
$\mathcal{Q}_4$	4.6585	0.4469	0.096

(b) Flow class $\mathcal{F}_{\text{ii}}$

	$\mathcal{I}_a$	$\mathcal{I}_b$	$\mathcal{I}_b$ / $\mathcal{I}_a$
$\mathcal{Q}_1$	4.8076	0.5232	0.109
$\mathcal{Q}_2$	14.7703	0.7101	0.048
$\mathcal{Q}_3$	3.1227	0.4493	0.144
$\mathcal{Q}_4$	6.7764	0.4309	0.064

(c) Flow class $\mathcal{F}_{\text{iii}}$

	$\mathcal{I}_a$	$\mathcal{I}_b$	$\mathcal{I}_b$ / $\mathcal{I}_a$
$\mathcal{Q}_1$	10.9531	0.6207	0.057
$\mathcal{Q}_2$	38.5549	0.8354	0.022
$\mathcal{Q}_3$	5.7535	0.4733	0.082
$\mathcal{Q}_4$	18.7430	0.5599	0.030

(d) Flow class $\mathcal{F}_{\text{iv}}$

	$\mathcal{I}_a$	$\mathcal{I}_b$	$\mathcal{I}_b$ / $\mathcal{I}_a$
$\mathcal{Q}_1$	8.6195	0.6662	0.077
$\mathcal{Q}_2$	11.7808	0.8811	0.075
$\mathcal{Q}_3$	6.7123	0.4301	0.064
$\mathcal{Q}_4$	10.2702	0.5645	0.055

Table 2. Comparison of $\mathcal{I}_a$ and $\mathcal{I}_b$ performance in seconds per dataset

	$\mathcal{F}_{\text{i}}$			$\mathcal{F}_{\text{ii}}$			$\mathcal{F}_{\text{iii}}$			$\mathcal{F}_{\text{iv}}$		
	$\mathcal{I}_a$	$\mathcal{I}_b$	$\mathcal{I}_b$ / $\mathcal{I}_a$	$\mathcal{I}_a$	$\mathcal{I}_b$	$\mathcal{I}_b$ / $\mathcal{I}_a$	$\mathcal{I}_a$	$\mathcal{I}_b$	$\mathcal{I}_b$ / $\mathcal{I}_a$	$\mathcal{I}_a$	$\mathcal{I}_b$	$\mathcal{I}_b$ / $\mathcal{I}_a$
$\mathcal{S}_{l,l,l}$	5.5666	0.5054	0.091	3.3699	0.5207	0.154	26.5421	0.6488	0.024	7.7278	0.6041	0.078
$\mathcal{S}_{l,l,m}$	5.6544	0.4989	0.088	2.6993	0.5134	0.190	12.9048	0.6137	0.048	9.4213	0.6354	0.067
$\mathcal{S}_{l,l,h}$	5.6304	0.5043	0.090	10.2625	0.5691	0.055	10.6876	0.6416	0.060	15.1030	0.6971	0.046
$\mathcal{S}_{l,h,l}$	5.8085	0.4940	0.085	1.5479	0.5193	0.335	31.5640	0.6483	0.021	6.8629	0.6285	0.092
$\mathcal{S}_{l,h,m}$	5.8462	0.4967	0.085	4.3644	0.5171	0.118	12.4716	0.6125	0.049	7.5289	0.6451	0.086
$\mathcal{S}_{l,h,h}$	5.9960	0.5000	0.083	10.3962	0.5720	0.055	11.0061	0.6555	0.060	16.1746	0.7054	0.044
$\mathcal{S}_{h,l,l}$	6.0348	0.4958	0.082	5.3076	0.5169	0.097	33.0351	0.6581	0.020	7.9743	0.5678	0.071
$\mathcal{S}_{h,l,m}$	5.9849	0.4984	0.083	1.6218	0.5066	0.312	13.2531	0.6295	0.047	1.6119	0.5764	0.358
$\mathcal{S}_{h,l,h}$	5.5507	0.5038	0.091	9.9075	0.5476	0.055	10.8158	0.6367	0.059	14.8312	0.7086	0.048
$\mathcal{S}_{h,h,l}$	5.9669	0.5192	0.087	1.5943	0.5079	0.319	31.0614	0.6595	0.021	6.1285	0.6327	0.103
$\mathcal{S}_{h,h,m}$	5.4860	0.4984	0.091	8.1835	0.5415	0.066	13.0176	0.6209	0.048	8.1223	0.6426	0.079
$\mathcal{S}_{h,h,h}$	5.6549	0.4814	0.085	10.1422	0.5555	0.055	10.6277	0.6359	0.060	14.8031	0.7003	0.047
$\mathcal{S}_t$	31.8727	0.4870	0.015	26.4032	0.4813	0.018	23.5278	0.4290	0.018	5.2046	0.5169	0.099
Avg	7.7733	0.4987	0.064	7.3693	0.5284	0.072	18.5011	0.6223	0.034	9.3457	0.6355	0.068

these flow classes focus on selecting SQL data before joining NoSQL data. Especially $\mathcal{Q}_2$ within $\mathcal{F}_{\text{iii}}$ shows that the average query time can be significantly reduced when the NoSQL data reconstruction is optimized.

When comparing the different datasets in Table 4 we again exclude $\mathcal{F}_{\text{i}}$ and $\mathcal{F}_{\text{ii}}$ since the optimization does not have an effect for these flow classes. For the other flow classes average results per dataset are presented. Again we see that the optimization typically results in a significant performance increase.

Discussion. After analyzing both experiments separately, we can draw some broad conclusions regarding the framework and optimizations. Firstly, Experiment 1 shows that a working prototype can be constructed based on the proposed framework. Furthermore, a temporary relation significantly improves the performance by an order of magnitude. The second experiment, conducted in

Table 3. Comparison of $\mathcal{I}_c$ and $\mathcal{I}_d$ performance in seconds per flow class

(a) Flow class $\mathcal{F}_{\text{iii}}$

	$\mathcal{I}_c$	$\mathcal{I}_d$	$\mathcal{I}_d$ / $\mathcal{I}_c$
$\mathcal{Q}_1$	13.5964	11.3169	0.832
$\mathcal{Q}_2$	34.1419	7.7161	0.226
$\mathcal{Q}_3$	9.2465	9.6599	1.045
$\mathcal{Q}_4$	7.0996	5.7576	0.811

(b) Flow class $\mathcal{F}_{\text{iv}}$

	$\mathcal{I}_c$	$\mathcal{I}_d$	$\mathcal{I}_d$ / $\mathcal{I}_c$
$\mathcal{Q}_1$	15.5227	11.8015	0.760
$\mathcal{Q}_2$	12.8297	8.9534	0.698
$\mathcal{Q}_3$	11.8088	10.2127	0.865
$\mathcal{Q}_4$	8.2041	6.1557	0.750

Table 4. Comparison of $\mathcal{I}_c$ and $\mathcal{I}_d$ performance in seconds per dataset

	$\mathcal{F}_{\text{iii}}$			$\mathcal{F}_{\text{iv}}$		
	$\mathcal{I}_c$	$\mathcal{I}_d$	$\mathcal{I}_d$ / $\mathcal{I}_c$	$\mathcal{I}_c$	$\mathcal{I}_d$	$\mathcal{I}_d$ / $\mathcal{I}_c$
$\mathcal{S}_{l,l,l}$	8.0736	1.3646	0.169	8.8565	1.7372	0.196
$\mathcal{S}_{l,l,m}$	9.6559	4.7486	0.492	6.7734	5.4149	0.799
$\mathcal{S}_{l,l,h}$	11.4418	11.5517	1.010	12.1739	13.2877	1.091
$\mathcal{S}_{l,h,l}$	9.1960	1.2582	0.137	11.1787	1.6435	0.147
$\mathcal{S}_{l,h,m}$	9.6356	4.4673	0.464	12.5762	5.2294	0.416
$\mathcal{S}_{l,h,h}$	11.3714	13.1159	1.153	12.8711	13.2781	1.032
$\mathcal{S}_{h,l,l}$	7.3243	4.2647	0.582	7.7039	3.2565	0.423
$\mathcal{S}_{h,l,m}$	11.3422	12.2677	1.082	13.8836	13.1399	0.946
$\mathcal{S}_{h,l,h}$	70.5085	20.8496	0.296	23.5845	22.5808	0.957
$\mathcal{S}_{h,h,l}$	11.1132	4.0471	0.364	6.6920	4.3302	0.647
$\mathcal{S}_{h,h,m}$	11.7446	12.8316	1.093	12.5065	13.6754	1.093
$\mathcal{S}_{h,h,h}$	35.7321	20.1385	0.564	22.7828	22.1383	0.972
$\mathcal{S}_t$	1.1351	1.0585	0.933	5.6044	0.9391	0.168
Avg	16.0211	8.6126	0.538	12.0913	9.2808	0.768

a larger-scale setting shows that the NoSQL data reconstruction strategy is indeed a successful optimization for many query classes. The final prototype, $\mathcal{I}_d$, while successful as a proof-of-concept, leaves space open for future improvements. In general, we conclude from our study the practical feasibility of the proposed query processing framework.

6 Conclusions

In this paper we have presented the first generic extensible framework for coordinated querying across SQL and NoSQL stores which eliminates the need for ad-hoc manual intervention of the user in query (re)formulation and optimization. An extensive empirical study demonstrated practical feasibility of the framework and the proposed implementation strategies.

The groundwork laid here opens many interesting avenues for further research. We close by listing a few promising directions. (1) We have just scratched the surface of implementation and optimization strategies. We give two suggestions for future work here. (a) We can study adaptations and extensions of indexing and caching mechanisms developed for RDF, a triple-based data model, and XML to more scalable implementations of F [13,18]. (b) Individual queries are often part of a longer-running collection. It would be certainly worthwhile to investigate strategies for multi-query optimization with respect to a dynamic query workload. (2) There is recent work in the community towards standardization of document query languages and their semantics [5,10,19]. An important interesting topic for further investigation is to coordinate our results with these emerging efforts (e.g., studying appropriate extensions or restrictions to NQP's).

Acknowledgments. We thank T. Calders and A. Serebrenik for their feedback.

References

1. Abiteboul, S., et al.: Web data management. Cambridge University Press (2011)
2. Aca, U., et al.: A graph model of data and workflow provenance. In: Proc. TAPP, San Jose, California (2010)
3. Agrawal, R., Somani, A., Xu, Y.: Storage and querying of e-commerce data. In: Proc. VLDB, Rome, pp. 149–158 (2001)
4. Atzeni, P., Bugiotti, F., Rossi, L.: Uniform access to non-relational database systems: The SOS platform. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 160–174. Springer, Heidelberg (2012)
5. Benzaken, V., Castagna, G., Nguyen, K., Siméon, J.: Static and dynamic semantics of NoSQL languages. In: Proc. ACM POPL, Rome, pp. 101–114 (2013)
6. Cattell, R.: Scalable SQL and NoSQL data stores. SIGMOD Record 39(4), 12–27 (2010)
7. Chong, E.I., Das, S., Eadon, G., Srinivasan, J.: An efficient SQL-based RDF querying scheme. In: Proc. VLDB, Trondheim, Norway, pp. 1216–1227 (2005)
8. Crockford, D.: The application/json media type for javascript object notation, JSON (2006), `http://www.ietf.org/rfc/rfc4627.txt`
9. Fletcher, G.H.L., Wyss, C.M.: Towards a general framework for effective solutions to the data mapping problem. J. Data Sem. 14, 37–73 (2009)
10. JSONiq, `http://jsoniq.org`
11. Litwin, W., Ketabchi, M., Krishnamurthy, R.: First order normal form for relational databases and multidatabases. SIGMOD Record 20(4), 74–76 (1991)
12. Ludäscher, B., Weske, M., McPhillips, T., Bowers, S.: Scientific workflows: Business as usual? In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 31–47. Springer, Heidelberg (2009)
13. Luo, Y., et al.: Storing and indexing massive RDF datasets. In: Virgilio, R., et al. (eds.) Semantic Search over the Web, pp. 31–60. Springer, Berlin (2012)
14. Management of external data (SQL/MED). ISO/IEC 9075-9 (2008)
15. Melnik, S., et al.: Dremel: Interactive analysis of web-scale datasets. PVLDB 3(1), 330–339 (2010)
16. Roijackers, J.: Bridging SQL and NoSQL. MSc Thesis, Eindhoven University of Technology (2012)
17. Sadalage, P.J., Fowler, M.: NoSQL distilled: A brief guide to the emerging world of polyglot persistence. Addison Wesley (2012)
18. Shanmugasundaram, J., et al.: A general technique for querying XML documents using a relational database system. SIGMOD Record 30(3), 20–26 (2001)
19. UnQL, `http://unql.sqlite.org`

Fast Multi-update Operations on Compressed XML Data

Stefan Böttcher, Rita Hartel, and Thomas Jacobs

University of Paderborn, Computer Science, Fürstenallee 11, 33102 Paderborn, Germany
{stb@,rst@,tjacobs@mail.}uni-paderborn.de

Abstract. Grammar-based XML compression reduces the volume of big XML data collections, but fast updates of compressed data may become a bottleneck. An open question still was, given an XPath Query and an update operation, how to efficiently compute the update positions within a grammar representing a compressed XML file. In this paper, we propose an automaton-based solution, which computes these positions, combines them in a so-called Update DAG, supports parallel updates, and uses dynamic programming to avoid an implicit decompression of the grammar. As a result, our solution updates compressed XML even faster than MXQuery and Qizx update uncompressed XML.

Keywords: updating compressed XML data, grammar-based compression.

1 Introduction

Motivation: XML is widely used in business applications and is the de facto standard for information exchange among different enterprise information systems, and XPath is widely used for querying XML data. However, efficient storage, search, and update of big XML data collections have been limited due to their size and verboseness. While compression contributes to efficient storage of big XML data, and many compressed XML formats support query evaluation, fast updates of compressed XML formats involve the challenge to find and to modify only those parts of an XML document that have been selected by an XPath query.

Background: We follow the grammar-based XML compression techniques, and we extend an XML compression technique, called CluX, by fast multi-update operations, i.e. operations that update multiple XML nodes selected by an XPath query without full decompression. Like the majority of the XML compression techniques, we assume that textual content of text nodes and of attribute nodes is compressed and stored separately and focus here on the compression of the structural part of an XML document.

Contributions: Our paper presents a new and efficient approach to simulate multi-update operations on a grammar-compressed XML document. That is, given a grammar G representing an XML document D, and given an update operation O to be performed on all nodes N of D selected by an XPath query Q, we can simulate O's modification of all nodes N on G without prior decompression. To the best of our knowledge, it is the first approach that combines the following properties:

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 149–164, 2013.

Our approach computes all update positions in G determined by Q in such a way that paths through the grammar to these update positions can be combined to a so-called Update DAG. This Update DAG can be used for updating multiple XML nodes at a time without full decompression of the grammar G. The Update DAG construction combines dynamic programming, a top-down evaluation of Q's main path, and a bottom-up evaluation of Q's filters. As our performance shows, this outperforms competitive query engines like QizX and MXQuery which work on uncompressed documents.

Paper Organization: For simplicity of this presentation, we restrict it to XML documents containing only element nodes. The next section introduces the idea of grammar based XML compression and of executing updates in parallel on such grammars. Based on these fundamentals, we describe our main contribution, the goal of which is to execute an XPath query on a given grammar and to compute the Update DAG that supports parallel updates. The evaluation of the entire approach is then shown in Section 4.

2 Fundamentals and Previous Work

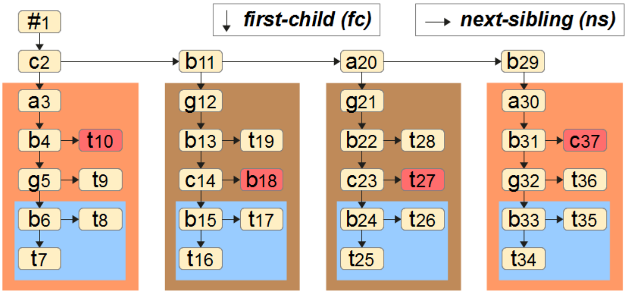


Fig. 1. Document tree of an XML document D with repeated matches of patterns

2.1 Sharing Similar Trees of XML Documents Using Grammars

Fig. 1 shows an example XML document D represented as a binary tree, where e.g. #'s first-child is c, the next-sibling of which is b. To distinguish multiple occurrences of node labels, we have numbered the nodes in pre-order. The simplest grammar-based XML compressors are those compressors that share identical sub-trees, such that the compressed grammar represents the minimal DAG of the XML tree [1]. These Approaches share identical sub-trees T in an XML document D by removing repeated occurrences of T in D, by introducing a grammar rule N→T, and by replacing each T by non-terminal N. Applying this approach to our example document D, the sub-tree b(t,t) is found with four matches, each of which is replaced by non-terminal A0.

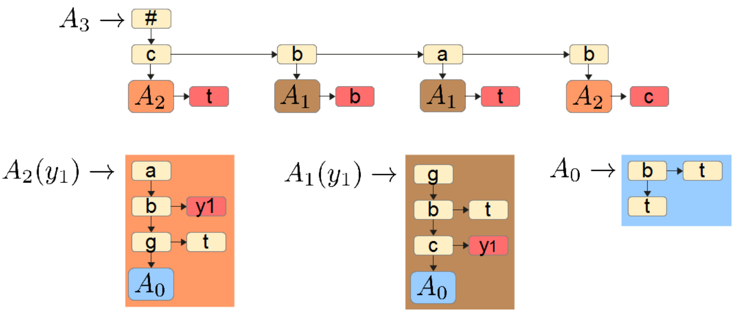


Fig. 2. Document of Fig. 1 with identical and similar sub-trees replaced by rule calls

However, a weakness of this approach is that only identical sub-trees can be compressed. In our example, the sub-trees rooted in nodes a3 and a30 differ only at the highlighted leaf nodes t10 and c37. By using approaches like CluX [2], BPLEX [3], or TreeRePAIR [4], we are able to compress those similar sub-trees by introducing parameterized grammar-rules. These grammar rules consist of the identical parts of the sub-trees and introduce parameters as placeholders for the different parts. Fig. 2 shows one possible resulting grammar, i.e. Grammar 2, represented as a set of trees. The similar sub-trees of Fig. 1 are replaced by non-terminals A1 and A2, which are the left-hand sides of new grammar rules in Grammar 1 containing y1 as a parameter.

```
A3     → #(c(A2(t),b(A1(b),a(A1(t),b(A2(c),ε)))),ε)
A2(y1) → a(b(g(A0,t),y1),ε)
A1(y1) → g(b(c(A0,y1),t),ε)
A0     → b(t,t)
```

Grammar 1: A grammar sharing similar sub-trees by using parameterized rules.

Each non-terminal Ai refers to exactly one grammar rule Ai(y1,y2,…,yn) →rhs(Ai), with rhs(Ai) being the right-hand side of that rule. We call yi a formal parameter (or just parameter). For a non-terminal expression Ai(t1,t2,…,tn) used in a right-hand side of a grammar-rule, we refer to each ti as an actual parameter. The grammars considered here are linear and straight-line. Linearity means that each parameter occurring on the left-hand side of a grammar rule appears exactly once in the right-hand side of that same rule. A grammar is straight-line, if the graph representing the rule calls is acyclic.

2.2 Using Grammar Paths to Identify Nodes

Each path to a selected node in an XML document D corresponds to exactly one grammar path (GP) in the grammar G producing D. Beginning with the start non-terminal of the grammar, this GP contains an alternating sequence of non-terminals Ai and index positions within rhs(Ai) to refer to a symbol, which is a non-terminal Ni calling the next grammar rule. It ends with the index of the symbol corresponding to the selected terminal in the last grammar rule of the GP.

For example, if we apply the XPath query **Q:=//a//b[./t]** to Grammar 1, one of the selected nodes can be described by GP1:=`[A3,3,A2,4,A0:1]`. Thus, GP1 describes a rule call to rhs(A2) at position 3 in rule A3 and a rule call to rhs(A0) at position 4 in rule A2. Finally, terminal b at position 1 in rhs(A0) is selected. A more formal definition of grammar paths, however omitting rule names, is given in [5].

2.3 Executing an Update-operation for a Given Grammar Path

Now suppose that we want to execute an update operation for a single given GP. As an example consider **GP1:=`[A3,3,A2,4,A0:1]`** and update operation relabelTo(z), which replaces the label b of the selected terminal to z. Clearly, just relabeling the first terminal in rhs(A0) would be wrong, since this terminal represents four nodes in the uncompressed XML document. One possible solution to this problem was presented in [6]. The idea is to first create a copy of each grammar rule occurring in GP1. Let Ai' represent the left-hand side non-terminals of these copied rules. Then, for each sub-sequence (Ai,k,Aj) in GP1, non-terminal Aj at position k in rhs(Ai') is replaced by Aj'. Additionally, for the last sub-sequence (An:k), the update operation (for example relabelTo(z)) is executed on symbol k in rhs(An'). Finally, the start rule is replaced by the copy of the start rule. Applying this strategy to GP1, yields Grammar 2 as a result. Note that the size of this grammar is not optimal and can be further compressed.

```
A3'     → #(c(A2'(t),b(A1(b),a(A1(t),b(A2(c),ε)))),ε)
A2(y1) → a(b(g(A0,t),y1),ε)        A2'(y1) → a(b(g(A0',t),y1),ε)
A1(y1) → g(b(c(A0,y1),t),ε)
A0      → b(t,t)                          A0' → z(t,t)
```

Grammar 2: Grammar 2 after applying relabelTo(z) to GP1=[A3,3,A2,4,A0 : 1].

2.4 The Concept of Parallel Updates on Grammars

Given an XPath query, usually a set of multiple GPs is selected. Thus, a desirable goal is to support executing updates on such a set of GPs in parallel and to keep the size of the grammar low. A first step towards a solution of this problem is to construct a prefix tree of the GPs [6]. This tree is constructed by introducing nodes with labels Ai for non-terminals Ai and directed edges with label k for sub-sequences (Ai,k,Aj) in the GPs to be updated. Furthermore, for sub-sequences (An:k), the tree-node created for An saves an entry k. The resulting graph is a tree, as equal prefixes in the grammar paths are combined, and since each grammar path begins in the start-rule. The resulting tree for the set of grammar paths selected by query **Q:=//a//b[./t]** is shown in Fig. 3(a), where edges to numbers represent entries saved in a node, i.e. positions of selected terminals.

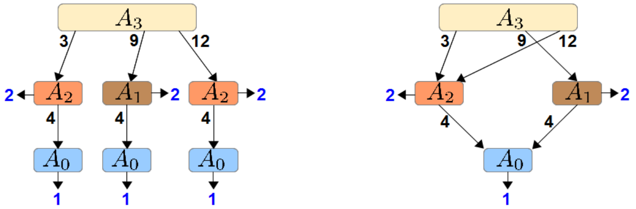


Fig. 3. a) Prefix Tree for query //a//b[./t] on Grammar 1, **b)** Corresponding Update DAG

The updates are now executed by walking top-down through the tree. Intuitively, the grammar rules on each tree-branch are isolated from the original grammar and then updated. That is, for each node of the tree visited, the corresponding grammar rule is copied. Let (Na,Nb) be an edge with label k and let label(Na)=Ai and label(Nb)=Aj respectively. With Ai' and Aj' being the non-terminals of the copied grammar rules, the symbol at position k in the grammar rule of Ai' is replaced by non-terminal Aj'. Finally, for an entry k saved in node Ni, the update is applied to the k-th symbol in rhs(Ai').

Although this approach works correctly, it induces a large overhead, since grammar rules are unnecessarily copied. For example, there are three equal nodes having label A0 in the tree of Fig. 3(a). Thus, copying the corresponding grammar rule once would have sufficed. The same holds for nodes with label A2. Formally, two leaf nodes are equal, if they save the same entries of selected terminals and have the same label, i.e. they correspond to the same non-terminal. Two inner nodes are equal, if they additionally have an identical number of outgoing edges with equal labels pointing to (recursively) equal child nodes. This finally brings us to the concept of parallel updates as introduced in [6]. Instead of looking at each grammar path for its own, we construct the (minimal) grammar path DAG from the prefix tree by combining equal nodes. This way, not only the size of the prefix tree is reduced, but additionally, we avoid unnecessary copying of grammar rules. In the context of executing update operations, we refer to this DAG as the Update DAG. The Update DAG for the given prefix tree of Fig. 3(a) is shown in Fig. 3(b). Executing the update operation relabelTo(z) then results in the more space saving Grammar 3. For a core XPath expression P, our approach supports the update operations P.relabelTo(z), P.deleteNodesAndTheirFirstChildSubtree() P.insertAsFirstChild(tree), and P.insertAsNextSibling(tree) on all selected nodes (More details are given in [6]).

A3' → #(c(A2'(t),b(A1(b),a(A1'(t),b(A2'(c),ε))))),ε)

~~A2(y1) → a(b(g(A0,t),y1),ε)~~ **A2'(y1) → a(z(g(A0',t),y1),ε)**

A1(y1) → g(b(c(A0,y1),t),ε) **A1'(y1) → g(z(c(A0',y1),t),ε)**

A0 → b(t,t) **A0' → z(t,t)**

Grammar 3: Grammar 2 after applying relabelTo(z) based on the Update DAG of Fig.3(b).

3 Construction of the Update DAG

3.1 Assumptions and Problem Definition

Let Q be an XPath query, O be an update operation, and G a straight-line linear grammar representing an XML document D. In the following, we assume that Q is an absolute query corresponding to *Core XPath* [7]. To simplify the explanations, we only consider non-nested relative filters excluding boolean operators. However, note that our software prototype obeys the complete Core XPath specification. Given these assumptions, the aim is to evaluate query Q on grammar G yielding the Update DAG to allow the execution of parallel updates.

3.2 Overview of Our Approach

Our algorithm directly computing the Update DAG consists of three main steps:

a. Given an XPath query Q, we follow the Looking Forward approach of [8], i.e., we rewrite Q in such a way that it consists of forward axes only. Additionally, we extract the filters with their current context nodes from the main path of Q.
b. Given the set of extracted filters, for each filter expression F, we construct a special bottom-up automaton to evaluate F on grammar G. As a result, for each filter expression F, we get the minimal grammar path DAG (called Filter DAG) containing all grammar paths to nodes in the document for which F is fulfilled.
c. As last step, we construct a top-down automaton for the main path of Q following the approach of [9]. To test, whether a filter is fulfilled in a node, we use the Filter DAGs constructed in Step b. The result of this step is the Update DAG.

To avoid an implicit decompression of the grammar in steps b and c, we follow and extend the idea of dynamic programming and hashing as introduced in [5].

3.3 Query Rewriting and Extraction of Filters

As a first step, we rewrite the given XPath query Q, such that it contains forward axes of the set {descendant, descendant-or-self, child, following-sibling, self} only. The example query Q=//a//b[./t] already contains forward axes only. From the rewritten query, we extract the filters from the main path, i.e., for each location step of the form ax::tst[pred] which is not part of another filter predicate itself, we extract tst[pred]. Furthermore, we keep references in the main path pointing to the extracted filters. For Q, this results in the main path **M=/descendant::a/descendant::b→F1** and the filter **F1=b[child::t]**.

3.4 Evaluation of Queries without Filters

Now let us first consider the evaluation of a query without filters. As the example query we use main path M, assuming filter F1 always evaluates to true. To evaluate

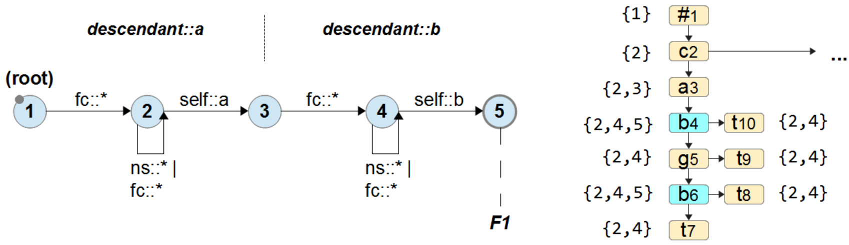


Fig. 4. a) Top-down automaton for main path M, **b)** Evaluation on the document tree of Fig. 1

the query, we extend our automaton-based top-down approach of [9] to work on grammars. It has the advantage that it is rather simple and allows us to use dynamic programming avoiding an implicit decompression of the grammar.

Constructing the Top-Down Automaton: The automaton for the main path of the query is constructed as presented in [9]. That is, each location step ax::tst can be described by an atomic automaton having transitions accepting events of the form binAx::tst, where binAx is a binary XPath axis first-child (fc), next-sibling (ns) or self. The main path then is the concatenation of these automata, as Fig. 4 (a) shows for the example query.

Evaluation on an Uncompressed Document Tree: The evaluation of such an automaton on uncompressed document trees works as described in [9]. The basic idea is to walk top-down in pre-order through the tree and to generate corresponding first-child, next-sibling and self-events. After visiting a first-child node, before continuing to the next-sibling of the parent node, a parent-event is generated, which resets the active automaton states to the states which were active in that parent node before. For example, for a tree b(t,t), the sequence (self::b, fc::*, self::t, parent::*, ns::*, self::t) is generated. Note that self-events are fired, as long as transitions can fire. A detailed description is given in [9]. Fig. 4 (b) sketches the evaluation (represented by sets of active states) of the automaton in Fig. 4 (a) that corresponds to Q's main path.

Evaluation on Grammars: As the evaluation of this top-down automaton so far only worked on uncompressed documents, we extended it to work on grammars and to directly compute the Update DAG. Our idea is to keep the automaton unchanged, but to introduce an additional module which traverses the grammar, generates grammar events, stores and recovers automaton state-sets and forwards some events to the automaton. Table 1 gives an overview of the algorithms involved in this module. Let V be the nodes and E be the edges of the DAG, both initialized with empty sets. The evaluation starts with calling procedure evalRule() for the start-rule of the grammar. For each (recursive) call of evalRule(), a DAG-node D is created, and entries(D) later-on store the positions within the currently visited grammar rule of terminals selected by the query. Each grammar rule is traversed top-down and corresponding events are generated. This works in the same way as for the uncompressed document, but with four new events ***terminal*** (replacing event self::*), ***nonTerminal***, ***actualParameter*** and ***formalParameter*** (c.f. Table 1). E.g., consider t(A(a),y1). For this expression, event-sequence (terminal(t,1), fc::*, nonTerminal(A,2), actualParameter, terminal(a,3), parent::*, formalParameter) is generated. Events fc::*, ns::* and parent::* are directly forwarded to the automaton. When discovering a terminal L, repeatedly an event self::L is forwarded to the automaton until no more transitions of the automaton can fire (line 17). Whenever the automaton accepts, we know that L is selected by the query, and we add L's position in the grammar rule to entries(D). Second, when a non-terminal Ak is found in the actual rule Ai, we recursively traverse the grammar rule for Ak, unless we can skip the traversal of Ak by dynamic programming as explained in the next sub-section. After processing Ak, we add an edge from

Table 1. Algorithm and events for top-down evaluation of a Grammar

```
(1)   procedure evalRule(non-terminal Nt): (DagNode node, list buffer)
(2)   {  D               = new DagNode;
(3)      label(D)        = Nt;
(4)      entries(D)      = empty set; //positions of selected terminals
(5)      actParamBuffer  = empty list;
(6)      formParamBuffer = empty list;
(7)      traverse and evaluate rhs(Nt) in pre-order and generate the
         events fc::*, ns::*, parent::*, terminal, nonterminal,
         formalParameter, actualParameter;
(8)      if (node-set V of DAG contains a node D′ equal to D)  D = D′;
(9)      else V = V ∪ D;
(10)     return (D,formParamBuffer);
(11)  }
(12)  event formalParameter
(13)     formParamBuffer.append(automaton.getActiveStates());
(14)  event actualParameter
(15)     automaton.setActiveStates(actParamBuffer.getAndRemoveHead());
(16)  event terminal(label L, int position)
(17)     do (automaton.fire(self::L)) while automaton.changesStates();
(18)     if (automaton.isAccepting()) entries(D).add(position);
(19)  event nonTerminal(label N, int position)
(20)     states = automaton.getActiveStates();
(21)     if (lemmaTable.contains((N,states)))         // skip calling N
(22)        (node,buffer) = lemmaTable.getValueForKey((N,states));
(23)     else                // cannot skip calling the production of N
(24)     {  (node,buffer) = evalRule(N);
(25)        key           = (N, states);
(26)        value         = (node, buffer);
(27)        lemmaTable.put(key,value);
(28)     }
(29)     edge         = (D,node);
(30)     label(edge)  = position;
(31)     E            = E ∪ edge;
(32)     actParamBuffer.prepend(buffer); //copy of buffer now list-head
```

the current DAG-node D to the DAG node returned by evalRule(Ak). Third, when a formal parameter is found, we must store the set of active states of the automaton, since we need these states later when continuing on the corresponding actual parameter of the calling grammar-rule (line 13). Intuitively, we freeze the automaton, until we later continue on the actual parameter of the calling rule. Fourth, when an actual parameter is found, we activate the state-set frozen in the automaton (line 15). We must know, which state-set we have to activate for continuing traversal on the actual parameter. Therefore, when previously processing the non-terminal of that actual parameter, we copy the state-sets to a list actParamBuffer (line 32). After the traversal of a grammar rule, procedure evalRule() checks, whether there is an equal DAG node already in set V, whereas equality is defined as in the section introducing parallel updates. Note, that this test can be done in time O(1) using hashing. As a result, evalRule() finally returns the root-node of the minimal Update DAG.

An example is shown in Fig. 5. While traversing rhs(A2), y1 is discovered and the active state-set of the automaton is stored in list formParamBuffer. After completing the traversal of rhs(A2), this state-set is returned by evalRule(A2) and prepended to actParamBuffer of evalRule(A3). Then, traversal continues at terminal t of rhs(A3), which is an actual parameter. Thus, the head of list actParamBuffer saving the state-set of the previously discovered parameter y1 is removed and activated in the automaton.

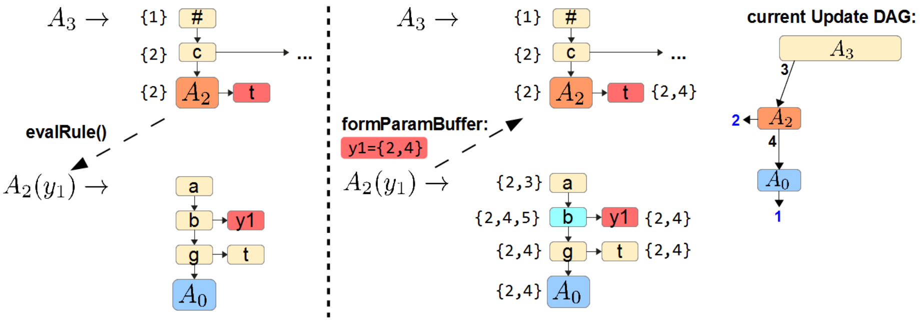


Fig. 5. a), b) Evaluation of the automaton of Fig. 4 (a) on Grammar 1

Optimization Using Dynamic Programming and Hashing: Using this approach, a problem still is that we would unnecessarily evaluate some grammar rules multiple times. To avoid this, we use dynamic programming and hashing for grammar-compressed XML documents as itwas introduced in [5] for bottom-up automaton-based evaluations. For our approach, we extended it to work for a top-down evaluation of the grammar, as well. We introduce a lemma hash-table mapping keys to values. A key is a tuple of a non-terminal label and state-set, whereas a value is a tuple storing the DAG-node which was created for that non-terminal and a list of state-sets holding one state-set for each formal parameter of that grammar rule. The observation is that when a rule was traversed before with the same set of states active in the automaton, then the subsequent traversal of that rule must produce an equal DAG-node and equal sets of automaton states for the formal parameters. The use of the lemma table is already implemented in event nonTerminal of Table 1. However, note that in the worst case, the lemma table does never permit skipping a grammar rule. In this case, we would still implicitly traverse the whole uncompressed document. A detailed analysis for a bottom-up traversal already was given in [5]. Similar results hold for the top-down traversal, too. However, our evaluations in Section 4 show that on all tested documents and queries we are faster using dynamic programming than without using it, reaching speed-ups up to a factor of 6.7.

3.5 Evaluation of Queries Having Filters

Our example query Q=//a//b[./t] has a filter, and we decomposed Q into the main path M=/descendant::a/descendant::b →F1 and the filter F1=b[child::t]. Therefore, when using the top-down approach of the last section for evaluating M, we somehow need to know for which terminals b on which grammar paths, the filter F1 is fulfilled. In our top-down query evaluation approach of [9] on uncompressed documents, this is done by using a top-down automaton for the filter, as well. An instance of this automaton is created each time a node b is selected by M, and this instance then is evaluated in parallel during top-down evaluation. However, this approach has the disadvantage that there may be several instances of a filter automaton evaluated in parallel, i.e. one for each b-node in this case. Furthermore, as the main path of the query can have more than one filter attached to any location step, the automaton processing that main path needs to store references to filter automata instances in its states. Therefore, we decided to use another approach. Before evaluating M, we evaluate the filters using bottom-up automata resulting in a DAG for each filter. Such a DAG saves all grammar paths to nodes for which the corresponding filter is fulfilled. Thus, for F1, this DAG saves all grammar paths to b-terminals having a child t. When afterwards evaluating the main path of the query, we can use the computed DAGs to decide, for which document node which filter is fulfilled. This approach has the major advantage that the automata for the top-down traversal are kept simple and that we can extend the idea of dynamic programming to consider filters.

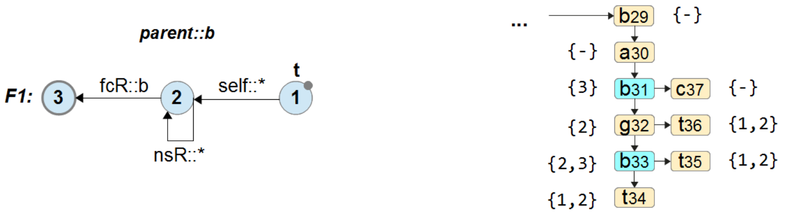


Fig. 6. a) Bottom-up automaton for F1=b[child::t], **b)** Evaluation on document tree of Fig. 1

Construction of Bottom-up Automata: To evaluate the extracted filters, we first construct a special bottom-up automaton for each filter reusing the ideas of [10]. The basic observation is that we can evaluate a location-path bottom-up from right to left, i.e., each expression tst1/ax::tst2 can be represented by the equivalent reverse expression tst2::axR/tst1, where axR is the reverse XPath axis of ax. For F1=b[child::t], we get t::parent/b. As in the top-down approach, each location step can then be expressed as an automaton using the events fcR::*, nsR::* and self::tst, with fcR being the reverse axis of first-child, nsR the reverse axis of next-sibling, and tst being a node test. Concatenating these automata for each location step then results in an automaton evaluating the whole filter expression. Fig. 6 (a) shows the resulting automaton for filter F1.

Evaluation on an Uncompressed Document Tree: A bottom-up automaton is evaluated on a tree by traversing the tree in a reversed post-order walk. Each time, when continuing evaluation at a leaf-node, a new instance of that automaton is created. When traversing the path from a leaf up to the root of the tree, corresponding events fcR::tst and nsR::tst are generated, with tst being the name of the node reached. For a leaf-node and for each node reached during traversal with label tst, an event self::tst is generated as long as transitions can fire. Note, that a transition with a label of the form axis::* can fire for any event axis::tst. Furthermore, for an event self::tst, the source states of firing transitions stay active for the same reasons as explained in [9]. The start-state of an automaton-instance is activated whenever a document node fulfilling the node name-test attached to the start-state of that automaton is found. As an optimization, when two leaf-to-root paths share a node, the two automata instances are unified to one instance at the first common node by computing the union of the active state sets. This way, sub-paths are traversed only once. Fig. 6 (b) visualizes the evaluation of the automaton for F1 on parts of the document tree of Fig. 1. The filter corresponding to the automaton is fulfilled in a document node n if and only if the automaton accepts in that node. In Fig. 6 (b), this holds for nodes b31 and b33, since they are the only b-nodes having a child with label t.

Evaluation on Grammars: The evaluation of each filter automaton on the grammar follows the idea of the top-down evaluation of the main path of the query. That is, we begin the traversal in the start rule and recursively traverse the rules of non-terminals, we find. The only difference is that grammar rules are processed bottom-up from right to left. This has the advantage that for a non-terminal expression Aj(p1,…,pn), the actual parameters pi are visited before Aj itself. For each actual parameter, the set of active states for every filter automaton is saved. When visiting non-terminal Aj afterwards, the traversal continues in rhs(Aj), which is processed in the same way bottom-up. When visiting a formal parameter yk, the state sets saved for actual parameter pk are activated in the automata. After finishing the traversal of rhs(Aj), processing continues in the calling grammar rule. A sketch of this is shown in Fig. 7 (a). Note that actual parameter t is visited before calling rhs(A2) and that the automaton states are transferred to y1 in rhs(A2). The Filter DAG is constructed in the same way as with the top-down approach. The (minimal) Filter DAG for filter F1 is shown in Fig. 7 (b).

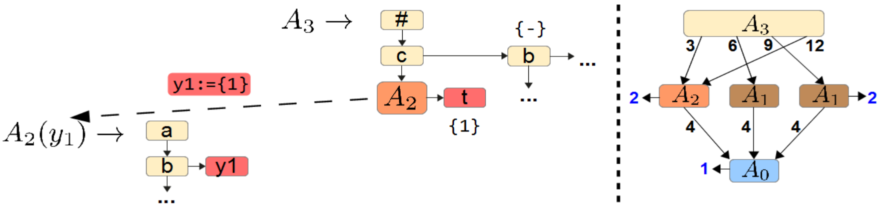


Fig. 7. a) Evaluation of F1 on Grammar 1, **b)** Resulting minimal Filter DAG

Optimization Using Dynamic Programming and Hashing: As for the top-down approach, we reuse the idea for the bottom-up evaluation of filters, as introduced in [5]. Again, we use a lemma hash-table mapping keys to values (one table for each filter). Each entry describes a rule-call from a non-terminal N occurring somewhere in the right-hand side of a production. A key is a tuple consisting of the non-terminal name of N and a list of state-sets. Each state-set at position i in the list describes the set of states which were active in the i-th actual parameter of N. The value is a tuple consisting of the DAG-node generated for traversing rhs(N) and the set of automaton-states, which were active after traversing that rule. The observation is that we can skip an additional traversal of rhs(N), when the automaton produces the same active state-sets for the actual parameters of the new occurrence of N. In this case, traversing the grammar rule again would produce the same DAG-node and the same active state-set.

As an example consider filter F1=b[child::t] evaluated on Grammar 1. Obviously, grammar rule rhs(A0) is traversed four times without dynamic programming. However, non-terminal A0 has no parameters, which means that evaluating the filter automaton on rhs(A0) always produces the same result. Thus, it is sufficient to traverse rhs(A0) only once. Furthermore, consider the grammar rule of A2. It is called two times from rhs(A3) by expressions A2(t) and A2(c). During the bottom-up evaluation of rhs(A3), the actual parameters t and c are visited before their non-terminal A2. Evaluating the automaton on terminals t and c yields state-sets {1,2} and {-} respectively. But this means, when processing rhs(A2) the automaton might produce different results, i.e. accept in different terminals and end-up in different active states after evaluating rhs(A2). This is, because the state sets computed in the actual parameters are used when visiting the formal parameter y1 in rhs(A2). Thus, we must not skip a second traversal through rhs(A2) here.

Note that we evaluate all filters of the query in parallel. Skipping the traversal of a grammar rule therefore is only allowed, if all lemma tables permit skipping. If only some of the lemma tables allow for skipping, we pause the automata of the corresponding filters such that only the automata which need to traverse that rule again are active.

Using the Filter DAGs during Evaluation: Now, having a Filter DAG for each filter, we must extend the top-down approach to use these DAGs. The general idea is to synchronously walk through the Filter DAGs while walking through the grammar. I.e., for the first call of evalRule(), we start in the root-nodes of the Filter DAGs. Then, for each recursive call to evalRule(), we follow the corresponding edges in the

Filter DAGs. This way, the top-down automaton can easily test, whether a filter is fulfilled in a currently visited terminal at position i, by checking whether the currently visited Filter DAG node of the corresponding filter stores an entry i. However, we also have to care about actual rule parameters for the following reason. Suppose, we have expressions N(b) and N(t) in a right-hand side of a grammar rule. Since we have filters and the actual parameters differ, different terminals may be selected in both rule-calls of rhs(N). Thus, we need to know, whether the filters for both calls of rhs(N) evaluate to true at the same positions. Exactly for this situation, we use the Filter DAGs. In a (minimal) Filter DAG, equal nodes have been combined to a single node. But this means, (only) when for both calls of rhs(N), in each Filter DAG, we are in the same Filter DAG node, the filters evaluate to true at the same positions. In this case, we can safely skip a second traversal of rhs(N). Thus, the decision to skip a traversal of a grammar rule also depends on the DAG-nodes currently visited and which were visited at the previous traversal of that rule. Therefore, we extend our lemma table of the top-down approach. A key-tuple additionally saves for each Filter DAG, the node which was active in that Filter DAG.

4 Evaluation

All tests were performed on an Intel Core2 Duo CPU P8800 @ 2.66 GHz. We used Java 1.7 (64 bit) with 2500 MB heap space assigned. As the first test document, we chose XMark (XM) which was generated using a scaling factor of 1.0 [11]. The second document is SwissProt (SP). To make our evaluations independent of the kind of text compression used, we removed all text- and attribute-nodes, resulting in documents of sizes 25.6 MB (XM) and 43.1 MB (SP), respectively. These documents were used as input for the query processors *QizX* [12] and *MXQuery* [13]. Furthermore, we used CluX to compress the documents, yielding grammars of size 1.15 MB (XM) and 1.74 MB (SP), respectively. These grammars were used as input to our algorithm using dynamic programming (*directUD*) and without using dynamic programming (*directUD no*). For a better comparison of the results, all documents were read from a RAM-disc. As running time, we measured the time, the algorithms spent in user mode on the CPU.

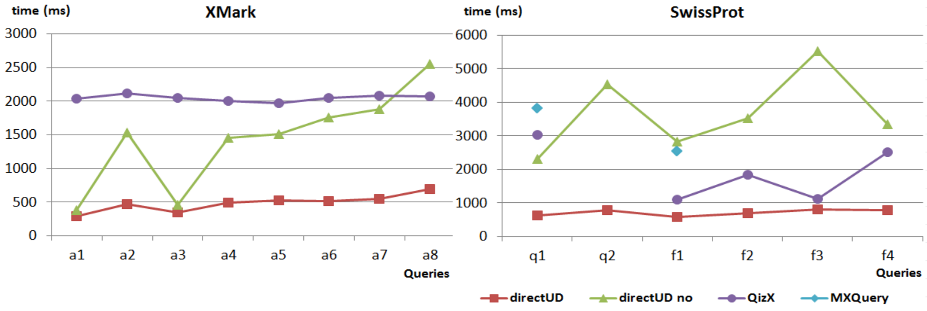


Fig. 8. a) Evaluation results on XMark document, **b)** on SwissProt document

Fig. 8 shows the evaluation results for both documents. The queries a1 to a8 executed on XM correspond to the XPath-A queries of the XPathMark benchmark [14]. For SP, queries q1 and f1 are designed such that they consist of few child axes only, whereas the other ones are more complex, having several following-sibling and descendant axes. Note that MXQuery currently does not support following-sibling axes and therefore was not executed on queries q2, f2 and f3. On the selected nodes by each query, a delete update operation was executed, removing these nodes including their first-child sub-trees. The times measured include both, the evaluation of an XPath query and the execution time of the update operation. Results not shown in the diagram were worse than 3 seconds for XM or worse than 6 seconds for SP, respectively. As both, Fig. 8 (a) and (b), show, our new approach outperforms QizX and MXQuery on each query. For the XMark document, we are about 4.2 times faster than QizX on average. However, when disabling dynamic programming, results get worse, such that QizX was faster than our algorithm for query a8. It has to be noted that query a8 has filters, such that our approach needs two runs through the grammar. Disabling dynamic programming results in implicitly decompressing that grammar twice. In this sense, our results show the benefit of using dynamic programming, being 3 times faster on average on the XMark document, when enabling it. In case of the SwissProt document, we benefit even more from dynamic programming, being up to 6.7 times faster when enabling it. Note that QizX was aborted after 60 seconds running on the rather complex query q2 having a rather high selectivity of 76,573 nodes, whereas our algorithm took less than one second.

5 Related Work

There are several approaches to XML structure compression which can be mainly divided into the categories: encoding-based, schema-based or grammar-based compressors. ***Encoding-based*** compressors (e.g.[15], [16], [17], XMill [18], XPRESS [19], and XGrind [20]) allow for a faster compression speed than the other ones, as only local data has to be considered in the compression as opposed to considering different sub-trees as in grammar-based compressors. ***Schema-based*** compressors (e.g. XCQ [21], Xenia [22], and XSDS [23]) subtract the given schema information from the structural information and only generate and output information not already contained in the schema information. XQzip [24] and the approaches [25] and [1] belong to ***grammar-based*** compression. They compress the data structure of an XML document by combining identical sub-trees. An extension of [1] and [24] is the BPLEX algorithm [3] that not only combines identical sub-trees, but recognizes similar patterns within the XML tree, and therefore allows a higher degree of compression. The approach presented in this paper, which is an extension of [2], follows the same idea. But instead of combining similar structures bottom-up, our approach searches within a given window the most promising pair to be combined while following one of three possible clustering strategies. Furthermore, in contrast to [5] and [26], that perform updates by path isolation only sequentially, our approach allows performing updates in parallel which takes only a fraction of time.

6 Summary and Conclusions

We have presented an approach to directly support updates on grammar-compressed big XML data. Given a grammar G representing an XML document D, and given an XPath query Q selecting nodes N of D and an update operation O to be performed on all these nodes N, our approach simulates this multi-update operation on G without full decompression of G. For this purpose, it computes the set of all grammar paths through G representing the nodes selected by Q, combines these paths into a small Update DAG, and then executes O in parallel on all the paths described by the Update DAG. As an advantage over other algorithms, there is no need to decompress the document and to compress it again afterwards. Additionally, by using the Update DAG, redundant modifications within the compressed grammar can be avoided, which increases the performance and keeps the size of the compressed XML document low. To further speed-up the execution of Q when computing update positions in G, we separate the top-down evaluation of Q's main path from the bottom-up computation of Q's filters, and we use dynamic programming for both, the top-down and the bottom-up computation. As a result, our solution outperforms other update processors like QizX and MXQuery working on uncompressed XML only up to a factor of 37 and more.

References

1. Buneman, P., Grohe, M., Koch, C.: Path Queries on Compressed XML. In: VLDB 2003, Berlin, Germany (2003)
2. Böttcher, S., Hartel, R., Krislin, C.: CluX - Clustering XML Sub-trees. In : ICEIS 2010, Funchal, Madeira, Portugal (2010)
3. Busatto, G., Lohrey, M., Maneth, S.: Efficient memory representation of XML documents. In: Bierman, G., Koch, C. (eds.) DBPL 2005. LNCS, vol. 3774, pp. 199–216. Springer, Heidelberg (2005)
4. Lohrey, M., Maneth, S., Mennicke, R.: Tree Structure Compression with RePair. In: DCC 2011, Snowbird, UT, USA (2011)
5. Fisher, D., Maneth, S.: Structural Selectivity Estimation for XML Documents. In: ICDE 2007, Istanbul, Turkey (2007)
6. Bätz, A., Böttcher, S., Hartel, R.: Updates on grammar-compressed XML data. In: Fernandes, A.A.A., Gray, A.J.G., Belhajjame, K. (eds.) BNCOD 2011. LNCS, vol. 7051, pp. 154–166. Springer, Heidelberg (2011)
7. Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing XPath queries. ACM Trans. Database Syst. 30 (2005)
8. Olteanu, D., Meuss, H., Furche, T., Bry, F.: XPath: Looking forward. In: Chaudhri, A.B., Unland, R., Djeraba, C., Lindner, W. (eds.) EDBT 2002. LNCS, vol. 2490, pp. 109–127. Springer, Heidelberg (2002)
9. Böttcher, S., Steinmetz, R.: Evaluating xPath queries on XML data streams. In: Cooper, R., Kennedy, J. (eds.) BNCOD 2007. LNCS, vol. 4587, pp. 101–113. Springer, Heidelberg (2007)
10. Benter, M., Böttcher, S., Hartel, R.: Mixing bottom-up and top-down xPath query evaluation. In: Eder, J., Bielikova, M., Tjoa, A.M. (eds.) ADBIS 2011. LNCS, vol. 6909, pp. 27–41. Springer, Heidelberg (2011)

11. Schmidt, A., Waas, F., Kersten, M., Carey, M., Manolescu, I., Busse, R.: XMark: A Benchmark for XML Data Management. In : VLDB 2002, Hong Kong, China (2002)
12. Axyana-Software: Qizx, `http://www.axyana.com/qizx`
13. MXQuery, `http://mxquery.org`
14. Franceschet, M.: XPathMark: An XPath Benchmark for the XMark Generated Data. In: Bressan, S., Ceri, S., Hunt, E., Ives, Z.G., Bellahsène, Z., Rys, M., Unland, R. (eds.) XSym 2005. LNCS, vol. 3671, pp. 129–143. Springer, Heidelberg (2005)
15. Zhang, N., Kacholia, V., Özsu, M.: A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML. In: ICDE 2004, Boston, MA, USA (2004)
16. Cheney, J.: Compressing XML with Multiplexed Hierarchical PPM Models. In: DCC 2001, Snowbird, Utah, USA (2001)
17. Girardot, M., Sundaresan, N.: Millau: an encoding format for efficient representation and exchange of XML over the Web. Computer Networks 33 (2000)
18. Liefke, H., Suciu, D.: XMILL: An Efficient Compressor for XML Data. In: SIGMOD 2000, Dallas, Texas, USA (2000)
19. Min, J.-K., Park, M.-J., Chung, C.-W.: XPRESS: A Queriable Compression for XML Data. In: SIGMOD 2003, San Diego, California, USA (2003)
20. Tolani, P., Haritsa, J.: XGRIND: A Query-Friendly XML Compressor. In: ICDE 2002, San Jose, CA (2002)
21. Ng, W., Lam, W., Wood, P., Levene, M.: XCQ: A queriable XML compression system. Knowl. Inf. Syst. (2006)
22. Werner, C., Buschmann, C., Brandt, Y., Fischer, S.: Compressing SOAP Messages by using Pushdown Automata. In: ICWS 2006, Chicago, Illinois, USA (2006)
23. Böttcher, S., Hartel, R., Messinger, C.: XML Stream Data Reduction by Shared KST Signatures. In: HICSS-42 2009, Waikoloa, Big Island, HI, USA (2009)
24. Cheng, J., Ng, W.: XQzip: Querying compressed XML using structural indexing. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 219–236. Springer, Heidelberg (2004)
25. Adiego, J., Navarro, G., Fuente, P.: Lempel-Ziv Compression of Structured Text. In: DCC 2004, Snowbird, UT, USA (2004)
26. Fisher, D., Maneth, S.: Selectivity Estimation. Patent WO 2007/134407 A1 (May 2007)

A Practical Approach to Holistic B-Twig Pattern Matching for Efficient XML Query Processing

Dabin Ding, Dunren Che, Fei Cao, and Wen-Chi Hou

Department of Computer Science
Southern Illinois University Carbondale
Carbondale IL 62901, USA
{dding,dche,fcao,hou}@cs.siu.edu

Abstract. Efficient twig pattern matching is essential to XML queries and other tree-based queries. Numerous so-called holistic algorithms have been proposed to process the twig patterns in XML queries. However, a more general form of twig patterns, called B-twig (or Boolean-twig), which allow arbitrary combination of all the three logical connectives, AND, OR, and NOT in the twig patterns, has not yet been adequately addressed. In this paper, a new approach accompanied with an optimal implementation algorithm is presented for efficiently processing B-twig XML queries. Experimental study confirms the viability and performance superiority of our new approach.

Keywords: Query processing, XML query, Tree pattern query, Boolean twig, Twig join, Tree pattern matching.

1 Introduction

XML as a *de facto* standard for data exchange and integration is ubiquitous over the Internet. Many scientific datasets are represented in XML, such as the Protein Sequence Database which is an integrated collection of functionally annotated protein sequences [2] and the scientific datasets at NASA Goddard Astronomical Data Center [1]. Moreover, XML is frequently adopted for representing meta data for scientific and other computing tasks. Efficiently querying XML data is a fundamental request to fulfill these scientific applications. In addition to examining the contents and values, an XML query requires matching the implied twig patterns against XML datasets. Twig pattern matching is a core operation in XML query processing. In the past decade, many algorithms have been proposed to solve the XML twig pattern matching problem. Holistic twig pattern matching has been demonstrated as a highly efficient overall approach to XML twig pattern computation.

Most of previous algorithms were designed to deal with *plain* twig patterns. However, queries in practical applications may contain all the three types of Boolean predicates, AND, OR, and NOT, resulting in a more general form of

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 165–174, 2013.

twig patterns that we used to call as All-twigs [4] and now tend to call as Boolean-Twigs [5]. The following is an example B-twig XML query (given in an XPath-like format) that involves all the three logical predicates, asking for the cars that are either "made by BMW after 2005" or "white but not a coupe":

/vehicle/car/[[made = 'BMW' AND year > 2005] OR [NOT[type = 'coupe'] AND color = 'white'].

As a practical example, the above query signifies that uniform and efficient support for AND, OR, and NOT is important for any XML query system.

We see only limited reported efforts made on holistically computing of B-twig pattern matches (a core operation in XML query evaluation). This paper reports our most recent result on this line of research, which is based on numerous years of efforts [4,5,7]. More specifically, we present a new and more efficient approach for B-twig pattern computing. With resort to minimum preprocessing, our new algorithm, *FBTwigMerge*, outperforms all prior related algorithms. With this new approach and the accompanying algorithm, we contribute to the community with a relatively complete family (this work, plus [5] and [7]) of approaches/algorithms for holistic B-twig pattern matching.

The remainder of the paper is organized as follows. Section 2 reviews related works. Section 3 provides preliminaries. Section 4 elaborates on our new approach. Section 5 presents our new algorithm, *FBTwigMerge*. Section 6 shows experimental results, and Section 7 concludes this paper.

2 Related Work

Twig pattern matching is a core operation in XML query processing. Holistic twig joins has been repeatedly demonstrated of performance superiority. In 2002, Bruno *et al.* [3] first proposed the so-called holistic twig join algorithm to XML twig queries, named *TwigStack*, whose main goal was to overcome the drawback of structural joins that usually generate large sets of unused intermediate results. *TwigStack* is not optimal when PC (parent-child) edges are involved in the twig patterns. Lu *et al.* [11] tried to make up this flaw and presented a new holistic twig join algorithm called *TwigStackList*, where a list structure is used to cache limited elements in order to identify a larger optimal query class. Chen *et al.* [6] studied the relationship between different data partition strategies and the optimal query classes for holistic twig joins. Grimsmo *et al.* [8] introduced effective filtering strategies into twig pattern matching and proposed algorithms that are worst-case optimal and faster in practice, which however address only the plain twigs. Lots of efforts have been reported on extending the holistic twig join approach to more general twigs such as Boolean twigs. Jiang *et al.* [10] made the first effort toward holistic computing of AND/OR-twigs. Yu *et al.* proposed a method for holistic computing of AND/NOT-twigs [15]. We [5] proposed and implemented the first holistic B-twig pattern matching algorithm called *BTwigMerge* (that allows arbitrary combinations of ANDs, NOTs, ORs in input twigs). The approach [5] resorts to normalization to regulate the combinations of AND, NOT, OR predicates in a B-twig, but normalization comes with a cost – extra processing and potential query expansion. We then designed an alternative algorithm, *DBTwigMerge* [7], that discards normalization (and its inherent drawbacks) and has "the theoretical beauty" of no preprocessing, but does not outperform in all cases. The new algorithm to be detailed in this paper combine the advantages of our prior algorithms and is overall the best.

In order to concentrate our energy on innovative holistic approaches, we have adopted the simply region encoding scheme for labeling the XML data elements. Besides region encoding, extended Dewey [12,13,14,9] has been shown more advantageous, and will be considered for adoption in the frameworks [5,7] we have proposed for holistic B-twig pattern matching in the future.

In contrast, our work focuses on more powerful, efficient, and innovative holistic computing schemes for arbitrary B-twig patterns. Our work is unique by itself – no other researchers have done the same kind of research – efficient and holistic B-twig pattern computing. Our new approach combines the benefits of our two prior approaches [5,7], and our new algorithm demonstrates superior performance in all cases. The subsequent sections elaborate on the details of our new approach and new algorithm.

3 Data Model and Tree Representation

We adopt the general perspective [3] that an XML database is a forest of rooted, ordered, and labeled trees where each node corresponds to a data element or a value, and each edge represents an element-subelement or element-value relation. We assume a simple encoding scheme, a triplet region code ($start, end, depth$), which is assigned to each data element in a XML document. Each node (element-type) in an input twig query is associated to an input *stream* of elements (represented using their respective region code) of the type. Our goal is to design a scheme to efficiently find all the mappings of an arbitrary B-twig onto an XML document (or dataset). An arbitrary B-twig may consist of the two general types of nodes, QNode and LgNode. Furthermore, a LgNode represents any of the following types of nodes: ANode, ONode and NNode [7]. For ease of presentation, we use the term "query node" to generally refer to any node (QNode or LgNode) that appears in a B-twig query. The answer to a B-twig query is a set of qualified twig instances (i.e., the embeddings of the twig pattern into the XML database). We assume the same *output model* for B-twigs as in [5].

4 A Practical Approach for for B-Twigs

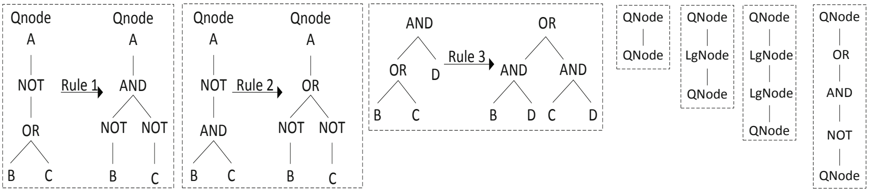
In this section, we present our new approach to holistic B-twig pattern matching. This approach is motivated from our two prior approaches [5,7]. We refer to our new approach as the "practical approach" as it combines the advantages of the two prior ones and outperforms both them in practice.

The key insight we obtained from our prior work [5,7] has two folds: normalization can effectively control combination complexity but may cause query expansion, and complete discard of normalization leads to very complex processing and inefficiency. The inspiration is to retain normalization but restrain to the extent that the net performance gain can be maximized. The bottom line is discard or minimize the normalization that causes major expansion on input B-twigs [5]. We particularly address the following three issues in the rest of this section: (i) identify the *favored forms* of B-twigs that yield efficient holistic processing; (ii) develop the preprocessing steps that transform an arbitrary B-twig into a desired form; (iii) design a new algorithm for efficiently (and holistically) evaluating the transformed B-twigs.

We call the favored/desired forms of B-twigs Well-Formed B-Twigs (or WFBTs for short) that tolerate many of the combinations that normalized forms [5] would not. The definition is given below.

Definition 1. *(Well-Formed B-Twig) A WFBT must satisfy: (1) not contain any superficially redundant edges such as AND-AND, OR-OR, NOT-NOT; (2) not involve three or more consecutive LgNodes between any two QNodes along any root-to-leaf paths, except for one special case of combination, OR-AND-NOT.*

WFBTs permit many combinations that would not be allowed by the normal forms, but are feasible to efficient algorithmic processing. Here we set the threshold "three" because we can pretty easily and efficiently deal with less than three consecutive LgNodes in a B-twig; for three (or more) consecutive LgNodes, we require to transform the input B-twig complying to WFBTs; the above-mentioned special case of combination, OR-AND-NOT, is already the favored pattern that the normalization discussed in [5] was purposefully seeking. When a B-twig is not recognized as a WFBT per Definition 1, it will be transformed into an equivalent WFBT in our new holistic approach. We retained three from the eight rules of [5] for transforming any non WFBT into an equivalent WFBT. These three rules, shown in Fig. 1(a), are adequate and necessary for obtaining WFBTs



(a) Transformation rules (b) Combinations

Fig. 1.

Theorem 1 (Completeness of Transformation). *By applying the three rules, every B-twig can be transformed into an equivalent WFBT per Definition 1.*

Proof. Let Q be an arbitrarily given B-twig. We leave out the following three trivial cases: (1) Q involves superficially redundant edges; (2) Q contains no more than three consecutive LgNodes between any two QNodes on any path; (3) Q contains the favored pattern of three consecutive LgNodes, OR-AND-NOT. We prove that any combination pattern other than OR-AND-NOT and involving three or more consecutive LgNodes on a path can be transformed into either (i) shorter patterns involving two or less consecutive LgNodes; or (ii) the favored OR-AND-NOT pattern through combined application of the three transformation rules.

With three consecutive LgNodes along a root-to-leaf path, there are totally 12 possible different combinations, including the favored one, OR-AND-NOT (details omitted due to space reason). We take AND-NOT-OR as an example for illustration: first, we apply Rule 1 to the NOT-OR sub-pattern, replacing the OR node with an AND node and pushing down the NOT node below the AND node (notice that the NOT node is duplicated by Rule 1); the above step results in an AND-AND edge along the path, and the two AND nodes shall be trivially merged into a singe one. If below at least one of NOT nodes immediately follows another LgNode, a further application of one of the three rules shall be carried out accordingly. Analogously for all other non-favored combinations of LgNodes. The correctness/equivalence of each performed transformation step is assured by the self-evident correctness of each of the three transformation rules. ■

To help perceive the implication of our WFBTs, we show, in Fig. 1(b), all the four distinct combination patterns (with limited length) of consecutive LgNodes that are allow in WFBTs. These patterns are *generic* in the sense that every LgNode may be replaced by either an AND, an OR, or a NOT node. From Fig. 1(b)), we can derive a total of 11 combinations that may immediately follow a QNode. This observation greatly facilitates the design of our algorithm for efficiently and holistically processing WFBTs.

Cost of Preprocessing. Transformation of an input B-twig into a WFBT comes with a cost. The cost consists of two parts: direct cost and indirect cost. The direct part counts the time consumed by the preprocessing itself and the indirect part counts any extra time spent by the query due to the alteration (including expected minor expansion). In practice, since the size of queries are usually small, the time consumed by preprocessing is usually marginal compared to the time spent on evaluating the query. When considering the indirect cost we need to carefully recheck our three transformation rules to see what side effects could be induced. Both Rule 1 and Rule 2 both introduce an additional NOT node, which is nothing because a negated query node can be processed as efficiently as a regular query node. Rule 3, while swapping AND and OR, may duplicate one (or more if applied repeatedly) child of an AND node (e.g., the D node of Rule 3 as shown in Fig. 1(a)). But the duplicated child just adds one more branch to the upper OR node, our optimized implementation for the OR predicate in a B-twig is able to identify such cases and evaluate duplicated query nodes *only once*. In all cases, no additional I/O cost is introduced, which is key to marginalize the preprocessing cost.

5 The *FBTwigMerge* Algorithm

In this section, we present our new algorithm, named *FBTwigMerge*, designed based on our above-sketched approach. Our new algorithm requires new mechanisms for efficiently evaluating WFBTs. Since our new approach shares the same overall framework as *BTwigMerge* [5], in the following we only give the important lower-level supporting mechanisms that are new or substantially different.

5.1 Novel Supporting Mechanisms for WFBTs

In order to holistically and efficiently computing the matches for WFBTs, we introduce the following important supporting mechanisms/functions:

- *edgeTest*: examines the containment relation between two elements, depicted as an edge of either '/' or '//'.
- *nEdgeTest*: examines a negated leaf QNode.
- *nNodeTest*: examines a negated non-leaf QNode.
- *ANodeTest*: evaluates a simple AND node (predicate) that has only QNodes as children (operands).
- *ONodeTest*: evaluates a simple OR node that has only QNodes as children.
- *SuperANodeTest*: evaluates an complex AND node that may bear other LgNodes as children.
- *SuperONodeTest*: evaluates an complex OR node that may bear other LgNodes as children.
- *SuperNNodeTest*: evaluates a complex NOT node that may bear another LgNode as its child.
- *hasExtension*: the key supporting function, examines whether an element indeed leads to a strict subtree match.

The *hasExtension* function, as the key supporting mechanism in the implementation of *FBTwigMerge*, examines whether there exists a strict subtree match. In Section 4, we summarized 11 distinct cases of combinations following a QNode, which must all be incorporated into *hasExtension*. This function in turn relies on a set of other, more fundamental supporting subroutines. Among them, *ANodeTest*, *NNodeTest*, *SuperANodeTest*, *SuperONodeTest*, and *SuperNNodeTest* are specially designed for *FBtwigMerge* that targets at WFBTs. In the following, due to space concern, we choose to elaborate on the algorithms of the three 'super' functions, which correspond to three generic cases involving AND, OR, and NOT, respectively.

Function *SuperANodeTest* handles the following three cases of potential combinations of LgNodes: QNode-ANode-QNode, QNode-ANode-ONode-QNode, and QNode-ANode-NNode-QNode per Definition 1. The ANode in the first case is a simple ANode while the ANodes in the other two cases are complex ANodes. Function *SuperONodeTest* deals with the following three potential combination patterns: QNode-ONode-QNode, QNode-ONode-ANode-QNode, and QNode-ONode-NNode-QNode. Algorithm 1 sketches the implementation of *Super-ONodeTest*, which (at line 7) calls function *SuperANodeTest* to deal with a potential complex ANode child. Function *SuperNNodeTest* deals with the following three potential combinations: QNode-NNode-QNode, QNode-NNode-ANode-QNode and QNode-NNode-ONode-QNode. The algorithms of *Super-ANodeTest* and *SuperNNodeTest* are omitted.

All these three new 'super' functions are called by the key supporting function, *hasExtension*, which has been substantially generalized (see Algorithm 2 and compare with [5]).

Algorithm 1. SuperONodeTest(ONode q, ONode q_i)

```
for all $q_j \in q_i$.getChildren() do
    if $q_j$.isQNode() then
        if $edgeTest(q, q_j)\&\&hasExtension(q_j)$ then
            return true
        end if
    else if  $q_j$.isANode() then
        if SuperANodeTest($q, q_j$) then
            return true;
        end if
    else if $q_i$.isNNode() then
        return $nNodeTest(q, q_j)$; //return anyway.
    end if
end for
```

Algorithm 2. hasExtension(QNode q)

```
for all $q_i \in q.getChildenList()$ do
    if $isQNode(q_i)$ then
        if $!(edgeTest(q, q_i)\&\&hasExtension(q_i))$ then
            return false;
        end if
    end if
    if $isANode(q_i)$ then
        if $!SuperANodeTest(q, q_i)$ then
            return false;
        end if
    else if $isONode(q_i)$ then
        if $!SuperONodeTest(q, q_i)$ then
            return false;
        end if
    else if $isNNode(q_i)$ then
        if $!SuperNNodeTest(q, q_i)$ then
            return false;
        end if
    end if
end for
return true;
```

5.2 Cost Analysis of *FBTwigMerge*

Relying on the core supporting function, *hasExtension*, our new algorithm, *FBTwigMerge*, probes through the input streams and seeks only for *exact* subtree matches. During the whole process, *FBTwigMerge* strictly differentiates PC edges from AD (ancestor-descendant) edges, generates (and pushes onto stacks) only useful elements that form optimal matches. As for I/O's, similar to *BTwigMerge* [5], *FBTwigMerge* never backtracks on any input stream. It (at the first phase) only sequentially scans through all relevant input streams once and produces path matches (intermediate results), and (at the second phase) reads the paths in (for resembling) and output final results at the end. So the total I/O sums up to $|input| + 3|output|$, which is overall linear to the sizes of input and output. We therefore can conclude that *FBTwigMerge* has worst-case optimal CPU and I/O costs, which are both linear to $|input| + |output|$.

6 Experiments

In this section we present the results of our experimental study and compare with related algorithms. We only compare with algorithms that also adopt region encoding instead of the Dewey encoding. The algorithms we selected for this study include *TwigStack* [3] (the first and representative algorithm for plain AND-only twigs), *GTwigMerge* [10] (the first and only known algorithm for AND/OR-twigs), *TwigStackList*¬ [15] (the first algorithm for AND/NOT-twigs), *BTwigMerge* [5] (the first and only known algorithm for normalized B-twigs), and *DBTwigMerge* (the first and only known direct algorithm for arbitrary B-twigs).

6.1 Experimental Setup

The platform of our experiments contains an Intel Core 2 DUO 2.2 GHz running Windows 7 System with 4GB memory. These algorithms were implemented and tested in JDK1.6. The region codes of elements and set of test queries were stored in external files on hard disk. To avoid potential bias of using a single dataset, all the experiments were conducted on two distinct data sets: the Protein Sequence Database (PSD) [2] and the Treebank (TB) dataset. The file size of PSD is 683MB and the size of TB is 82MB. We carefully considered the following factors, topology, distribution of logical nodes, selection ratio, and designed four sets of total 40 queries. Each set represents a distinct class of B-twigs: Set 1 consists of AND-only twigs; Set 2 consists of AND/OR-twigs; Set 3 consists of AND/NOT-twigs; and Set 4 consists of full B-twigs. This study focused on the key performance metric — CPU cost. Scalability test was also conducted on varied dataset sizes (i.e., multiples of a selected base dataset).

6.2 Performance Results

We conducted four sets of tests respectively with the PSD and TB datasets. For compactness, with each pair of corresponding tests (respectively with PSD and TB), we over-stack the performance charts in the same figure (left y-axis reads out the running time of the queries on TB, and the right y-axis reads out the running time of queries on PSD), and only indicate applicable algorithms in the performance charts.

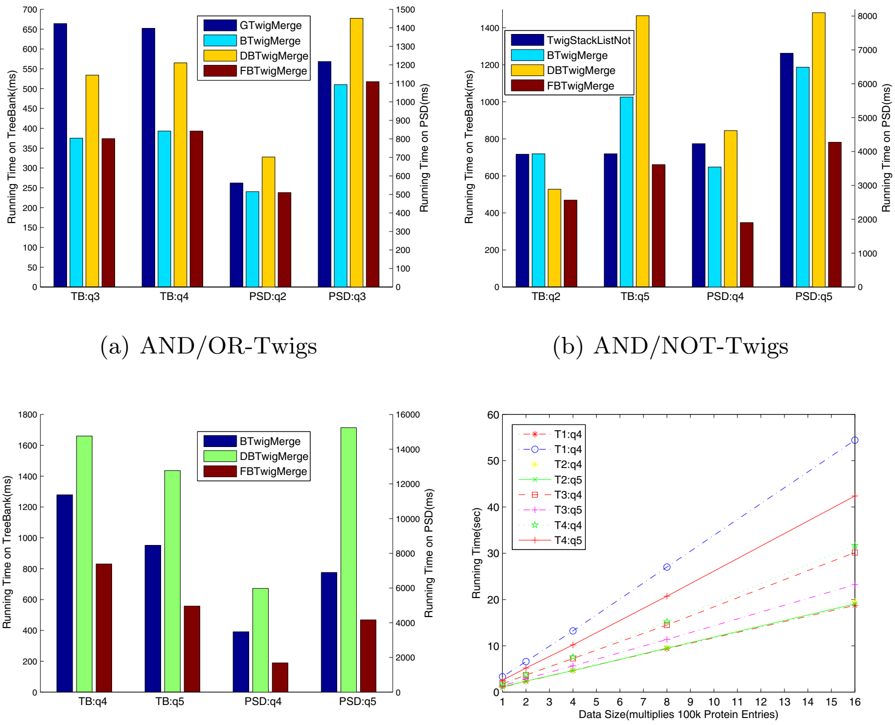
Test 1: With Plain Twigs. All the 6 tested algorithms show comparable performance on plain twigs (performance figure omitted for space reason).

Test 2: with AND/OR-Twigs. As we can see in Fig. 2(a) regarding the 4 applicable algorithms, *BTwigMerge* and *FBTwigMerge* have similar performance and both perform the best. On TB, *GTwigMerge* perform the worst, while on PSD, *DBTwigMerge* perform the worst. The reason why *DBTwigMerge* perform worse than *BTwigMerge* and *FBTwigMerge* lies the followings: first, *DBTwigMerge* incorporates logical nodes into its iterations, which takes up more time; second, the normalization/preprocessing step can actually simplify the input AND/OR-twigs to a certain degree, depending the pattern of the input queries, which is to the disadvantage of *DBtwiMerge* (in contrast, *BTwigMerge* and *FBTwigMerge* do not take logical nodes into their main iterations).

Test 3: with AND/NOT-Twigs. As shown in Fig. 2(b), *FBTwigMerge* perform the best in almost all the queries tested, due to the mentioned advantage of preprocessing on input B-twigs. For q5 of both PSD and TB, *DBTwigMerge* performs the worst because these queries contains a relatively large number of LgNodes, led to many extra processing iterations that do not exist in the other algorithms. For q5, *BTwigMerge* performs the worst. The reason that causes *BTwigMerge* to slowdown is the twig expansion rendered by its normalization resulted from repeated pushing-down of NOTs. In contrast, unfavorable normalization is reduced to the minimum in *FBTwigMerge*, which helps circumvent the drawback of *BTwigMerge*.

Test 4: with Full B-Twigs. For AND/OR/NOT-twig (or full B-twig) queries, only *BTwigMerge*, *DBTwigMerge*, and *FBTwigMerge* are applicable. The experiment result is plotted in Fig. 2(c). *DBTwigMerge* performs the worst with this set of queries due to the same reason as we explained with the AND/OR-twig and AND/NOT-twig queries. *FBTwigMerge* is the overall winner.

Scalability Test. Scalability test for *FBTwigMerge* is done with varied (increased) dataset sizes. For this purpose, 8 most complicated queries from our test sets were chosen and evaluated over the PSD dataset that was resized at 5 different scales. The i^{th} dataset contains $2^{i-1} \times 100k$ Protein entries, and the sizes of the datasets range from 276MB to 4.31GB. The results of scalability



(a) AND/OR-Twigs

(b) AND/NOT-Twigs

(c) AND/OR/NOT-Twigs

(d) Scalability Test

Fig. 2. Running Results

test are plotted in Fig. 2(d), from where we can see that our *FBTwigMerge* demonstrates nearly linear scalability of CPU time with all tested queries.

7 Summary

Holistic twig join is a critical operation in XML query processing. All three types of logical predicates, AND, OR, and NOT, are equally important to general XML queries. In this paper, we presented a novel and efficient approach for holistic computing of B-twig patterns and detailed its implementation. Our new algorithm, *FBTwigMerge*, outperforms all previous algorithms though not very significantly but pretty *consistently* in all test cases. The consistent out-performance of *FBTwigMerge* makes it the best option to practical XML query systems.

References

1. NASA Goddard Astronomical Data Center (ADC) Scientific Dataset in XML, `http://xml.coverpages.org/nasa-adc.html`
2. XML Data Repository, `http://www.cs.washington.edu/research/xmldatasets/`
3. Bruno, N., Koudas, N., Srivastava, D.: Holistic twig joins: Optimal XML pattern matching. In: Proc. of SIGMOD, pp. 310–321 (2002)
4. Che, D.: Holistically processing XML twig queries with AND, OR, and NOT predicates. In: Proc. of Infoscale, p. 53 (2007)
5. Che, D., Ling, T.W., Hou, W.-C.: Holistic Boolean twig pattern matching for efficient XML query processing. In: IEEE TKDE, vol. 24, pp. 2008–2024 (November 2012)
6. Chen, T., Lu, J., Ling, T.W.: On boosting holism in XML twig pattern matching using structural indexing techniques. In: Proc. of SIGMOD, pp. 455–466 (2005)
7. Ding, D., Che, D., Hou, W.-C.: A direct approach to holistic boolean-twig pattern evaluation. In: Liddle, S.W., Schewe, K.-D., Tjoa, A.M., Zhou, X. (eds.) DEXA 2012, Part I. LNCS, vol. 7446, pp. 342–356. Springer, Heidelberg (2012)
8. Grimsmo, N., Bjørklund, T.A., Hetland, M.L.: Fast optimal twig joins, vol. 3, pp. 894–905. VLDB Endowment (September 2010)
9. Izadi, S.K., Härder, T., Haghjoo, M.S.: S3: Evaluation of tree-pattern XML queries supported by structural summaries. Data Knowl. Eng. 68, 126–145 (2009)
10. Jiang, H., Lu, H., Wang, W.: XML query efficiency: Efficient processing of twig queries with OR-predicates. In: Proc. of SIGMOD, pp. 59–70 (2004)
11. Lu, J., Chen, T., Ling, T.W.: Efficient processing of XML twig patterns with parent child edges: A look-ahead approach. In: Proc. of CIKM, pp. 533–542 (November 2004)
12. Lu, J., Ling, T.W., Chan, C.-Y., Chen, T.: From region encoding to extended dewey: On efficient processing of XML twig pattern matching. In: Proc. of VLDB, pp. 193–204 (August 2005)
13. Lu, J., Ling, T.-W., Yu, T., Li, C., Ni, W.: Efficient processing of ordered XML twig pattern. In: Andersen, K.V., Debenham, J., Wagner, R. (eds.) DEXA 2005. LNCS, vol. 3588, pp. 300–309. Springer, Heidelberg (2005)
14. Soh, K.H., Bhowmick, S.S.: Efficient evaluation of NOT-twig queries in tree-unaware relational databases. In: Yu, J.X., Kim, M.H., Unland, R. (eds.) DASFAA 2011, Part I. LNCS, vol. 6587, pp. 511–527. Springer, Heidelberg (2011)
15. Yu, T., Ling, T.-W., Lu, J.: Twigstacklist-: A holistic twig join algorithm for twig query with not-predicates on XML data. In: Li Lee, M., Tan, K.-L., Wuwongse, V. (eds.) DASFAA 2006. LNCS, vol. 3882, pp. 249–263. Springer, Heidelberg (2006)

Representing MapReduce Optimisations in the Nested Relational Calculus

Marek Grabowski[1], Jan Hidders[2], and Jacek Sroka[1]

[1] Institute of Informatics, University of Warsaw, Poland
{gmarek,sroka}@mimuw.edu.pl
[2] Delft University of Technology, The Netherlands
a.j.h.hidders@tudelft.nl

Abstract. The MapReduce programming model is recently getting a lot of attention from both academic and business researchers. Systems based on this model hide communication and synchronization issues from the user and allow processing of high volumes of data on thousands of commodity computers. In this paper we are interested in applying MR to processing hierarchical data with nested collections such as stored in JSON or XML formats but with restricted nesting depth as is usual in the nested relational model. The current data analytics systems now often propose ad-hoc formalisms to represent query evaluation plans and to optimize their execution. In this paper we will argue that the Nested Relation Calculus provides a general, elegant and effective way to describe and investigate these optimizations. It allows to describe and combine both classical optimizations and MapReduce-specific optimizations. We demonstrate this by showing that MapReduce programs can be expressed and represented straightforwardly in NRC by adding syntactic short-hands. In addition we show that optimizations in existing systems can be readily represented in this extended formalism.

1 Introduction

MapReduce (MR) is a programming model developed at Google to easily distribute processing of high volumes of data on thousands of commodity computers. Systems based on this model hide communication and synchronization issues from the user, while enforcing a simple yet powerful programming style which is influenced by functional programming. MR is being successfully applied [6] on Web scale data at Google processing centers. After Google published the paper explaining the idea behind MR, an open source version, named Hadoop [2], was created and started to be widely used by both universities for research and companies like Yahoo, Twitter and Facebook to process their data.

Even though the MR model makes writing distributed, data-driven software a lot easier than with older technologies like MPI or OpenMP, for many applications it is too general and low level. This forces the developers who want to process large collections of data to deal with multiple concerns. Additionally to dealing with the problem they are trying to solve, they have to struggle with

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 175–188, 2013.

implementing and optimizing typical operations. As happened many times in history, e.g., with compilers or relational databases, it is better to separate concerns by introducing a higher level, more declarative language, better suited for specifying tasks for data analytics or scientific data processing, and at the same time more amenable to optimization. This is a topic of intensive study at the time of this writing and many systems are being built on top of implementations of MR ranging from data analytics Pig [10], data warehousing Hive [16], through workflow systems like Google's Sawzall [15], to graph processing systems like Pregel [14].

An effort is also undertaken on finding the best formal model of MR computations and their cost that would allow to reason which algorithms can be efficiently expressed in MR, and which cannot and why. Several attempts were made to define cost model, which is easy to use, understand and allows to reason on MR programs efficiency. One of the more successful ideas is a notion of *replication rate* introduced by Afrati, Ullman et al. [1], who count the number of excessive data transfers and claim that it is a good metric of MR efficiency since it deals with the bottleneck of typical MR programs — the network efficiency. In another work by Karloff et al. [12] a notion of MR expressive power was researched and a correspondence showed between MR framework and a subclass of PRAM.

In this paper we are interested in applying MR to processing hierarchical data with nested collections such as stored in JSON or XML formats but with restricted nesting depth as is usual in the nested relational model. We show that the Nested Relational Calculus (NRC) [3], a core language that describes a small but practically and theoretically interesting class of queries and transformations over nested relational data, is a good formalism to reason about MR programs. We demonstrate this by showing how some of the most important higher-level MR languages can be expressed in NRC with added syntactic short-hands. In addition we show that NRC provides a general, elegant and effective way to describe and investigate optimizations used in existing systems.

A similar attempt was made by Lämmel [13], who expressed the MR framework in t he Haskell language. Our approach is more formal and slightly stronger, as we are using a more minimal and formally defined calculus which is not Turing complete.

2 The Nested Relational Calculus: NRC

In this paper we are interested in operations on large datasets, mainly representing large collections of data. For simplicity we will restrict ourselves to one collection type: bags. This collection type has more algebraic properties than lists, which can be exploited for optimization purposes, and the natural notion of grouping is easily represented in terms of bags. Moreover, sets can be introduced straightforwardly by means of a **set** operator which removes duplicates from a bag. We denote bags by generalizing set enumeration, and so $\{a, a, b\}$ denotes the bag that contains a twice and b once. The bag union is denoted as $\uplus$

and is assumed to be additive, e.g., $\{a, b\} \uplus \{b, c, c\} = \{a, b, b, c, c\}$. In this paper we consistently use curly brackets { } to denote bags, not sets as usual.

Our data model will be essentially that of the nested relational model. Allowed data types are (1) *basic types* which contain *atomic values* being constants from some domain, which is assumed here to include at least integers and booleans, and to allow equality tests (2) named tuples with field names being strings and values being of any of allowed data types and (3) bag types which describe finite bags containing elements of a specify type. The instances of types will be called both *data* and *values* interchangeably.

For the purpose of optimization we will focus on a limited set of operators and language constructs that is a variant of the NRC. We will use the following syntax for our variant of NRC:

$$\begin{aligned} E ::= {} & C \mid X \mid \emptyset \mid \{E\} \mid E \uplus E \mid \langle K : E, \ldots, K : E\rangle \mid E.K \mid \\ & \mathbf{set}(E) \mid F(E) \mid \{E \mid X \in E, \ldots, X \in E, E, \ldots, E\} \mid E \approx E. \end{aligned}$$

We discuss the constructs in the order of appearance. The first one is C which describes denotations of constant atomic values. The nonterminal X stands for variables. The following three constructs form the basic bag operations, i.e., the empty bag constructor, the singleton bag constructor and the additive bag union. Usually also a typing regime is introduced with the calculus to ensure well-definedness and for example require that the bag union is only applied to bags of the same type, but for brevity we omit this and refer the reader to [17,7].

Next we have the tuple creation and tuple projection. In this paper we will be working with named tuples. The nonterminal K stands for field names and it must hold that all field names are distinct. The types that describe tuples, called *tuple types*, are defined as partial functions form field names to types, i.e., $\langle id : 1\rangle$ and $\langle value : 1\rangle$ have different types, but $\langle 1 : int, 2 : bool\rangle$ and $\langle 2 : bool, 1 : int\rangle$ have the same type. We will sometimes omit the column names. Notice that we allow the empty tuple as well, which will be called *unit* and denoted $\langle\rangle$.

The next construct is the **set** operator, which removes all duplicates from a bag. The F represents *user defined functions* (UDFs). We require that UDFs come with a well defined semantics, being a partial function from all possible values to all possible values, and are describe by a type. The Nested Relational Calculus as presented here is parametrized with the set of user-defined functions F, which is denoted as NRC^F.

The bag comprehensions of the form $\{e \mid x_1 \in e_1, \ldots, x_n \in e_n, e'_1, \ldots, e'_m\}$, also known from some functional programming languages [18] and query evaluation languages [9], returns a bag, which is constructed in the following way: starting with $x_1 \in e_1$ and going to the right we iterate over the given collection (here e_1) and assign one element at the time to the given variable (here x_1) in a nested-loop fashion. In the body of the innermost loop the expressions $e'_1, \ldots, e'_m$ are evaluated, and if any of their values is **false** then no output is generated, otherwise e is evaluated and its value added to the returned collection. Finally the construct of the form $e_1 \approx e_2$ denotes the equality between values and returns **true** if e_1 and e_2 evaluate to the same values, and **false** otherwise. The equality between tuples or bags is defined in a standard way.

Below we define the semantics for the NRC, which starts with the definition of substitution of variable x with expression e' in expression e, denoted as $e_{[x/e']}$. Its formal semantics can be defined by induction on the structure of e as follows:

$$\dfrac{}{c_{[x/e]} = c} \qquad \dfrac{}{x_{[x/e]} = e} \qquad \dfrac{x \neq y}{x_{[y/e]} = x} \qquad \dfrac{}{\emptyset_{[x/e]} = \emptyset} \qquad \dfrac{}{\{e\}_{[x/e']} = \{e_{[x/e']}\}}$$

$$\dfrac{}{(e_1 \uplus e_2)_{[x/e']} = e_{1[x/e']} \uplus e_{2[x/e']}}$$

$$\dfrac{}{\langle \kappa_1 : e_1, \ldots, \kappa_n : e_n \rangle_{[x/e]} = \langle \kappa_1 : e_{1[x/e]}, \ldots, \kappa_n : e_{n[x/e]} \rangle} \qquad \dfrac{}{e.\kappa_{[x/e']} = e_{[x/e']}.\kappa}$$

$$\dfrac{}{\mathbf{set}(e)_{[x/e']} = \mathbf{set}(e_{[x/e']})} \qquad \dfrac{}{f(e)_{[x/e']} = f(e_{[x/e']})} \qquad \dfrac{}{\{e \mid\}_{[x/e']} = \{e_{[x/e']} \mid\}}$$

$$\dfrac{}{\{e \mid x_1 \in e_1, \Delta\}_{[x_1/e']} = \{e \mid x_1 \in e_{1[x_1/e']}, \Delta\}}$$

$$\dfrac{x_1 \neq y \qquad \{e \mid \Delta\}_{[y/e']} = \{e'' \mid \Delta''\}}{\{e \mid x_1 \in e_1, \Delta\}_{[y/e']} = \{e'' \mid x_1 \in e_{1[x_1/e']}, \Delta''\}}$$

$$\dfrac{\{e \mid \Delta\}_{[y/e']} = \{e'' \mid \Delta''\}}{\{e \mid e_1, \Delta\}_{[y/e']} = \{e'' \mid e_{1[x_1/e']}, \Delta''\}} \qquad \dfrac{}{(e_1 \approx e_2)_{[x/e']} = (e_{1[x/e']} \approx e_{2[x/e']})}$$

Now we define the NRC semantics, i.e., the relation $e \Rightarrow v$ which denotes that expression e returns value v. It is defined in the following way:

$$\dfrac{}{c \Rightarrow c} \qquad \dfrac{}{\emptyset \Rightarrow \emptyset} \qquad \dfrac{e \Rightarrow v}{\{e\} \Rightarrow \{v\}} \qquad \dfrac{e \Rightarrow \{v_1, \ldots, v_n\}, e' \Rightarrow \{v'_1, \ldots, v'_n\}}{e \uplus e' \Rightarrow \{v_1, \ldots, v_n, v'_1, \ldots, v'_n\}}$$

$$\dfrac{e_1 \Rightarrow v_1, \ldots, e_n \Rightarrow v_n}{\langle \kappa_1 : e_1, \ldots, \kappa_n : e_n \rangle \Rightarrow \langle \kappa_1 : v_1, \ldots, \kappa_n : v_n \rangle} \qquad \dfrac{e \Rightarrow \langle \kappa_1 : v_1, \ldots, \kappa_n : v_n \rangle}{e.\kappa_i \Rightarrow v_i}$$

$$\dfrac{e \Rightarrow \{v_1, \ldots, v_n\}}{\mathbf{set}(e) \Rightarrow \cup_{i=1}^{n} \{v_i\}} \qquad \dfrac{f(v) \Rightarrow v' \qquad e \Rightarrow v}{f(e) \Rightarrow v'} \qquad \dfrac{e_1 \Rightarrow \mathbf{false}}{\{e \mid e_1, \Delta\} \Rightarrow \emptyset}$$

$$\dfrac{e_1 \Rightarrow \{v_1, \ldots, v_n\} \qquad \forall_{i=1}^{n} (\{e \mid \Delta\}_{[x_i/v_i]} \Rightarrow v'_i)}{\{e \mid x_1 \in e_1, \Delta\} \Rightarrow \uplus_{i=1}^{n} v'_i} \qquad \dfrac{e_1 \Rightarrow \mathbf{true} \qquad \{e \mid \Delta\} \Rightarrow v}{\{e \mid e_1, \Delta\} \Rightarrow v}$$

$$\dfrac{e \Rightarrow v}{\{e \mid\} \Rightarrow \{v\}} \qquad \dfrac{e \Rightarrow v \qquad e' \Rightarrow v' \qquad v \neq v'}{e \approx e' \Rightarrow \mathbf{false}} \qquad \dfrac{e \Rightarrow v \qquad e' \Rightarrow v}{e \approx e' \Rightarrow \mathbf{true}}$$

where we let $\oplus_{i=1}^{n} S_i$ denote $S_1 \oplus \ldots \oplus S_n$. Observe that the result of an expression is defined iff the expression contains no free variables.[1]

[1] Note that we do not require that the free variables of the substituted expression are not bound after the substitution, since we only substitute values.

3 MapReduce

MapReduce (MR) is a programming model for heavily distributed software, which hides most of the complexity coming from parallelism. The system handles communication, synchronization and failure recovery, while the user is responsible only for writing the program logic in the form of Map and Reduce functions. The working of the system is described in detail in [6], below we give only an outline of its design.

The input of a MR routine is a collection of key-value pairs. The main assumption is that the collection is too big to fit into the memory of a machine, so it is necessary to distribute the computations over multiple machines. MR is built on top of a Distributed File System (DFS) (for example the Google File System [11]) and takes advantage of its architecture, as the input is stored in DFS, thus it is divided into blocks, spread and replicated throughout a cluster.

The first stage of an MR routine is the *Map* phase. In the ideal case there is almost no communication needed during this phase, as the system tries to process the data on machines that already store it, which is often feasible thanks to the replication. In real life some of the data may need to be sent, but we chose to ignore it, as it is too low level for our model and the amount of necessary communication depends on many hard to predict factors, like cluster configuration and its load. After the map phase, the user can choose to run the combine phase, which takes all the data from a single mapper for a single key and runs a UDF on such a collection. As this phase is designed only to improve efficiency not the expressive power of the model, e.g. some frameworks may choose not execute *Combine* functions, we chose to skip this phase all together.

The next stage of MR is opaque for the user and is called the *Shuffle* phase. It consists of grouping all the *Map* outputs which have the same intermediate key and sending the data to the machines on which *Reduce* functions will be run. In practice there is a sort order imposed on the intermediate keys, and sometimes also on the grouped data, but we choose to ignore the order and work on bags instead, since the order is rarely relevant at the conceptual level of the transformation, i.e., users usually think about their collections as bags and do not care about the specific ordering. This is the stage where communication and synchronization takes place, and opaqueness of it makes the reasoning about MR routines easier. In some implementations of MR the user can specify a partitioner, which is responsible for distributing the data between machines running the reducers. Note that this behavior may be modeled using secondary-key grouping, as it is required that all datagrams with the same intermediate key end up on the same machine.

The last phase is called the *Reduce* phase, and it consists of independent executions of the *Reduce* function, each on a group of values with the same intermediate-key, and produces a collection of *result* key-value pairs.

It is possible to feed a Reducer from multiple different mappers, as long as the *Shuffle* phase can group the outputs of all the mappers together. In such a case, the intermediate data from all mappers is treated identically and is merged together for shuffling. Furthermore, often MR routines are pipelined together, making the output of one routine, an input of another one. In such a case product-keys of the former, become input-keys of the latter. Sometimes by MR we do not mean a MR construct, but a computation model consisting of MR routines and ability to connect those routines to form a DAG. Such computation model is parametrized with a class of allowed Map and Reduce UDFs. It should be clear from the context which meaning of MR we have in mind.

We chose a simplified version of MR, without ordering, intermediate *Combine* phase, Partitioning etc., as this is the model appearing most often in the literature. Furthermore some of our simplifications do not impact the expressive power of the model, which what we are interested in this paper. Those simplifications may turn out to be too strong in the future, to work with some low-level query optimizations, but they proved to be appropriate for the optimizations we are considering in this paper.

4 Defining MapReduce in NRC

We proceed by showing that the MR framework can be defined using NRC constructs described in the previous section. Here we want the reader to note that both **Map** and **Reduce** phases apply *Map* and *Reduce* UDFs to the data. In the general case functions passed as arguments to **Map** and **Reduce** can be arbitrary, as long as they have following types: $\langle k : \alpha_1, v : \beta_1\rangle \to \{\langle k : \alpha_2, v : \beta_2\rangle\}$ and $\langle k : \alpha_2, vs : \{\beta_2\}\rangle \to \{\langle k : \alpha_3, v : \beta_3\rangle\}$, respectively. Here in the rest of the paper we use the short names k, v and vs for *key*, *value* and *values* respectively, to make the presentation shorter.

Note that in our definition of NRC there are no functions per se, so in stead we use expressions with free variables. To denote expressions with abstracted variables we use the λ notation, e.g. if e is a NRC expression with a single free variable x, with the semantics well defined for x of a type α with a result type β, then $\lambda x.e$ is a function of type $\alpha \to \beta$.

The **Map** routine, where the first argument is a *Map* UDF, and D is a phase input of type $\{\langle k : \alpha_1, v : \beta_1\rangle\}$, can be written in NRC as:

$$\textbf{Map}\quad [\lambda x.e_{map}](D) = \{z \mid y \in D, z \in e_{map[x/y]}\}.$$

Note that we assume that the result of $\lambda x.e_{map}$ is a collection and the **Map** flattens the result, hence the output of the **Map** has the same type as $\lambda x.e_{map}$.

The *Shuffle* phase at the conceptual level essentially performs a grouping on the key values. It can be expressed in NRC as:

$$\textbf{Shuffle}(D) = \{\langle k : x, vs : \{z.v \mid z \in D, z.k \approx x\}\rangle \mid x \in \textbf{set}(\{y.k \mid y \in D\})\},$$

The result of the *Shuffle* phase is a collection of key-collection pairs of all values grouped by keys.

The *Reduce* phase gets the output of the *Shuffle* phase, which is of type collection of key-collection pairs, and is responsible for producing the result of the whole MapReduce routine. It can be formulated in NRC as:

$$\textbf{Reduce} \quad [\lambda x.e_{red}](D) = \{z \mid y \in D, z \in e_{red[x/y]}\},$$

Having defined all phases of MR separately, we can define the whole **MR** syntactic short-hand:

$$\textbf{MR} \quad [\lambda x.e_{map}][\lambda x.e_{red}](D) = \textbf{Reduce}\ [\lambda x.e_{red}](\textbf{Shuffle}\,(\textbf{Map}\ [\lambda x.e_{map}](D))),$$

The extension of NRC with the **MR** syntactic short-hand construct will be referred to as **NRC-MR**. Our definition or MR construct allows only one mapper. It is easy to generalize it to handle multiple mappers, as long as they have a common output type. To do so we need to feed the **Shuffle** with the union of all mapper outputs.

We define a **MR** program informally as a workflow described by a directed acyclic graph that ends with a single node and where each internal and final node with n incoming edges as associated with (1) an **MR** step with n mappers and (2) an ordering on the incoming edge. Moreover, the start nodes with no incoming edges are each associated with a unique input variable. When executing this program the data received through the ith input edge is fed to the ith mapper of that node. Note that our definition of **MR** program is valid for standard data processing, where **MR** is a top-level language. Sometimes, e.g. in case of workflows with feedback or graph algorithms, there is an additional level of programming needed on top of **MR** which introduces recursion. Our model can be seen as a formalization of **MR** programs where the dataflow does not depend on the actual data, which is the case for most of database queries.

Theorem 1. *Any MR program where the* **MR** *steps use as mappers and reducers functions expressible in NRC*^{F} *can be expressed in NRC*^{F}.

Proof. Indeed any MR routine using functions from F can be written in NRC using the **MR** shorthand, as showed above. Composing the routines into a DAG is equivalent to nesting the expressions for corresponding routines, in NRC.

5 Higher-Level MapReduce-Based Languages

Recently, high-level languages compiled to MR are receiving a lot of attention. Examples of those attempts are Facebook's Hive [16] – a Hadoop based data warehouse with SQL-like query language, Yahoo!'s Pig Latin [10] – a data analysis imperative language with SQL-like data operations, and Fegaras's MRQL [8] – an extension of SQL, which allows rnesting and in which MR is expressible in the same sense as in the NRC. In this section we review the ideas and operators from those languages and also provide an overview of the types of optimizations their implementations include.

5.1 Hive QL

Hive [16], is designed to replace relational databases, so some of its features like data insertion are orthogonal to our perspective. The Hive compilation and execution engine uses knowledge of the physical structure of the underlying data store. Since we abstract from a concrete physical representation of the data, we concentrate on the Hive Query Language (Hive QL). Its SQL-based syntax allows subqueries in the *FROM* clause, equi-joins, group-by's and including MR code. Hives does a handful of optimizations which are applied while creating the execution plan, including: (1) combining multiple *JOIN*s on the same key into a single multi-way join,(2) pruning unnecessary columns from the data, (3) performing map-side *JOIN*s when possible and (4) tricks based on knowledge of the physical structure of the underlying data store.

5.2 Pig Latin

Pig Latin [10] is a query language for the Pig system. It is business intelligence language for parallel processing huge data sets. The data model of Pig is similar to the one in this paper, with nesting and data types like tuples, bags and maps. Unlike other languages discussed here, Pig Latin is not declarative. Programs are series of assignments, and similar to an execution plan of a relational database. The predefined operators are iteration with projection *FOREACH-GENERATE*, filtering *FILTER*, and *COGROUP*. The *COGROUP*s semantics is similar to a *JOIN* but instead of flattening the product, it leaves the collections nested, e.g., **COGROUP** *People* **BY** *address*, *Houses* **BY** *address* returns a collection of the type: $\langle \mathit{address}, \{\mathit{People\ with\ given\ address}\}, \{\mathit{Houses\ with\ given\ address}\}\rangle$. It is easy to see that *GROUP* is a special case of *COGROUP* where the input is a single bag, and *JOIN* is a *COGROUP* with a flattened result. In addition Pig Latin also provides the user with some predefined aggregators, like *COUNT*, *SUM*, *MIN* etc., which we skip in our work since their optimized implementation is a research topic on its own and requires the inclusion of the *Combine* phase.

On the implementation and optimization side, the Pig system starts with an empty logical plan and extends it one by one with the user-defined bags, optimizing the plan after each step. Pig generates a separate MR job for every *COGROUP* command in the expression. All the other operations are split between those *COGROUP* steps and are computed as soon as possible, i.e. operations before the first *COGROUP* are done in the very first Map step, and all the others in the reducer for the preceding *COGROUP*.

5.3 MRQL

MRQL is a query language designed by Fegaras et al. [8] as a declarative language for querying nested data, which is as expressive as MR. The language is also designed to be algebraic in the sense that all the expressions can be combined in arbitrary ways. MRQL expressions are of the form:

$$\begin{array}{l}\textbf{select}\ e\ \textbf{from}\ d_1\ \textbf{in}\ e_1, d_2\ \textbf{in}\ e_2, \ldots, d_n\ \textbf{in}\ e_n\\ [\textbf{where}\ \mathit{pred}]\ [\textbf{group\_by}\ p' : e'\ [\textbf{having}\ e_h]]\ [\textbf{order\_by}\ e_o\ [\textbf{limit}\ n]]\end{array}$$

where e's denote nested MRQL expressions.

What is the most interesting in *MRQL* from our perspective, is not the language itself since it is similar to SQL, but the associated underlying physical algebraic operators. The main two operators are *groupBy* and flatten-map/*cmap* as known from functional programming languages. Those are the two operators which are needed to define the MR operator. Our approach is similar to Fegaras's, but in contrast we have one language for both query specification and query execution. An *MRQL* program is first rewritten to a simpler form if possible, and then an algebraic plan is constructed. The operators in such plan are *cmap*s, *groupBy*s and *join*s. Possible optimizations are:
- combining *JOIN*s and *GROUP BY*s on the same key into a single operation, (1) choosing an optimal *JOIN* strategy depending on the data, (2) fusing cascading *cmap*s, (3) fusing *cmap*s with *joins*, (4) synthesizing the *Combine* function. In section 7 we show all those optimizations, except the last one, can be done in NRC. The last one is skipped because for the sake of simplicity we do not include *Combine* functions in our execution model.

6 Defining Standard Operators in NRC

In this section we take a closer look on the operators found in the higher-level MR languages described in the previous section. We show how operators from those three languages can be defined in NRC. This illustrates that our framework generalizes the three considered languages.

6.1 SQL Operators

We start from the standard SQL operators, which form the basis of the three analyzed languages. For the sake of clarity, we sometimes abuse the notation, to make things clearer, e.g. we avoid the *key-value* pair format in MR expressions, if the keys are not used in the computation. In this section we assume that x is an element from collection X, wherever X denotes a collection.

The first and the most basic operator is the projection. Assuming that X has the type $\{\alpha\}$, $\alpha = \langle \cdots \rangle$, and π is some function, usually a projection on a subtuple of α, we have the following equivalent formulas:

$$\begin{aligned}\textbf{SQL, MRQL}: &\ \textit{SELECT}\ \pi(x)\ \textit{FROM}\ X,\\ \textbf{Pig Latin}: &\ \textit{FOREACH}\ X\ \textit{GENERATE}\ \pi(x)\\ \textbf{NRC}: &\ \{\pi(x) \mid x \in X\}\\ \textbf{MR-NRC}: &\ \textbf{MR}\ [\lambda x.\{\pi(x)\}][\lambda x.\textbf{id}_R](X),\end{aligned}$$

Here $\textbf{id}_R = \{\langle x.k, y\rangle \mid y \in x.vs\}$ and is an "identity" reducer.

The second operator is filtering. Assuming that X is a collection of type $\{\alpha\}$ and $\varphi : \alpha \to \textbf{boolean}$, the formulas for filtering are as follows:

$$\begin{aligned}\textbf{SQL, MRQL}: &\ \textit{SELECT}\ *\ \textit{FROM}\ X\ \textit{WHERE}\ \varphi(x),\\ \textbf{Pig Latin}: &\ \textit{FILTER}\ X\ \textit{BY}\ \varphi(x),\\ \textbf{NRC}: &\ \{x \mid x \in X, \varphi(x)\},\\ \textbf{MR-NRC}: &\ \textbf{MR}\ [\lambda x.\{y \mid y \in \{x\}, \varphi(y)\}][\lambda x.\textbf{id}_R](X),\end{aligned}$$

In some cases it is more efficient to apply projection or filtering in the *Reduce phase.* Corresponding alternative MR versions for these cases are $\mathbf{MR}\,[\lambda x.\{x\}][\lambda x.\{\pi(y) \mid y \in x.vs\}](X)$ and $\mathbf{MR}\,[\lambda x.\{x\}][\lambda x.\{y \mid y \in x.vs, \varphi(y)\}](X)$, respectively. Note that moving those operators, as well as the *cmap*, between the mapper and the reducer is a straightforward rewrite rule.

The third and most complex construct we are interested in is *GROUP BY.* Below we assume that X has the type $\{\alpha\}$, with $\alpha = \langle \dots, \kappa : \beta, \dots \rangle$ and π_κ is a projection of type $\alpha \to \beta$.

$$\begin{aligned}
\textbf{SQL, MRQL} &: \textit{SELECT} \ * \ \textit{FROM}\ X\ \textit{GROUP\_BY}\ \pi_\kappa(x),\\
\textbf{Pig Latin} &: \textit{GROUP}\ X\ \textit{BY}\ \pi_\kappa(x),\\
\textbf{NRC} &: \{\{x \mid x \in X, \pi_\kappa(x) \approx y\} \mid y \in \mathbf{set}(\{\pi_\kappa(x) \mid x \in X\})\},\\
\textbf{MR-NRC} &: \mathbf{MR}\ [\lambda x.\{\langle k : \pi_\kappa(x), v : x\rangle\}][\lambda x.x\}](X).
\end{aligned}$$

6.2 Pig Latin, HiveQL and MRQL Operators

We move to operators unique to higher-level MR languages, viz., *cmap* from the physical layer of MRQL and similar to the comprehension operator, which is based on a *map* construct instead of a *cmap*; and the *COGROUP* from Pig Latin, which can be seen as a generalization of *GROUP* and *JOIN* operators. HiveQL does not add new operators on top of the SQL ones. We leave for future work all forms of the *ORDER BY* operator.

First let us look at the $\textit{cmap}(f)X$ from MRQL, which is based on the concat map well known from the functional languages. The typing of this construct is as follows: $X : \{\alpha\}$, $f : \alpha \to \{\beta\}$ and $\textit{cmap}(f) : \{\alpha\} \to \{\beta\}$. It can be easily expressed in NRC as $\{y \mid x \in X, y \in f(x)\}$.

Provided that f does not change the key, i.e., $f : \langle k : \alpha, v : \beta\rangle \to \langle k : \alpha, v : \gamma\rangle$ such that $f(\langle k : e, v : e'\rangle).k \equiv e$, we can move the application of f between the mapper and the reducer. The efficiency of either choice depends on whether f inflates or deflates the data. In the first case it is better to have it in the reducer, in the second case in the mapper:

$$\begin{gathered}
\mathbf{MR}\ [\lambda x.f(x)][\lambda x.\mathbf{id}_R](D) \equiv\\
\mathbf{MR}\ [\lambda x.\{x\}][\lambda x.\{z \mid y \in x.vs, z \in f(\langle k : x.k, v : y\rangle)\}](D)
\end{gathered}$$

This rule can be generalized such that it allows f to be split into a part that is executed in the mapper and a part that is executed in the reducer.

The *COGROUP* is the only operator in which the nested data model is crucial. The syntax of *COGROUP* in Pig Latin is:

$$\textit{COGROUP}\ X\ \textit{BY}\ \pi_\kappa(x),\ Y\ \textit{BY}\ \pi_\iota(y),$$

where $X : \{\alpha\}$, $Y : \{\beta\}$, $\pi_\kappa : \alpha \to \gamma$, and $\pi_\iota : \beta \to \gamma$. The NRC expression for computing *COGROUP* has the form:

$$\begin{aligned}
&\{\langle a, \{x \mid x \in X, \pi_\kappa(x) \approx a\}, \{x \mid x \in Y, \pi_\iota(x) \approx a\}\rangle\\
&\quad \mid a \in \mathbf{set}(\{\pi_\kappa(x) \mid x \in X\} \uplus \{\pi_\iota(x) \mid x \in Y\})\}.
\end{aligned}$$

The MR routine for computing the *COGROUP* has the form (note the use of multiple mappers):

$$\begin{aligned}\mathbf{MR}\ \ &[\lambda x.\{\langle k : \pi_\kappa(x), v : \chi_1^2(x)\rangle\}, \lambda x.\{\langle k : \pi_\iota(x), v : \chi_2^2(x)\rangle\}]\\ &[\lambda x.\langle k : x.k, v : \langle\{z \mid y \in x.vs, z \in y.v_1\}, \{z \mid y \in x.vs, z \in y.v_2\}\rangle\rangle](X, Y),\end{aligned}$$

where $\chi_i^j(x)$ stands for $\langle v_1 : \emptyset, \dots v_i : \{x\}, \dots v_j : \emptyset\rangle$. Here the mapper creates tuples with two data fields, the first of which corresponds to the first input, and the second to the second input. Mappers put the input data in the appropriate field as a singleton bag and an empty bag in all other fields. The reducer combines those fields into the resulting tuple. The *COGROUP* is the first operator which spans through both the map and the reduce phase. This is the reason why it is important in the Pig query planner.

7 NRC Optimizations of Higher-Level Operators

In this section we show how optimizations described in [8,10,16] can be represented as NRC rewrite rules. We briefly recall the optimizations mentioned in section 5: (1) pruning unnecessary columns from the data, (2) performing map-side *JOIN*s when possible, (3) combining multiple *JOIN*s on the same key, (4) combining *JOIN*s and *GROUP BY*s on the same key, to a single operation, (5) fusing cascading *cmap*s, (6) fusing *cmap*s with *JOIN*s, (7) computing projections and filterings as early as possible in the intervals between *COGROUP*s – fusing projections and filterings with *COGROUP*s. In the order of appearance, we describe the optimizations and present NRC rewrite rules corresponding with each given optimization. By $=$ we denote syntactic equality, while by $\equiv$ we denote semantic equivalence.

Pruning unnecessary columns strongly depends on the type of the given data and expression. Pruning columns can be easily expressed with well-known NRC rewrite rules, and so we will assume we are working with expressions that project unused columns away as soon as possible.

The map-side join (2) is a technique of computing the join in the mapper, when one of the joined datasets is small enough to fit into a single machine's memory, thus its applicability is data-dependent. It is achieved by replacing the standard reduce-side *COGROUP*-based *JOIN* operator by applying the following rewrite rule:

$$\begin{aligned}\mathbf{MR}\ \ &[\lambda x.\{\langle k : \pi_\kappa(x), v : \chi_1^2(x)\rangle\rangle\}, \lambda x.\{\langle k : \pi_\iota(x), \chi_2^2(x)\rangle\}]\\ &[\lambda x.\{\langle k : x.k, v : \theta(y_1, z_1)\rangle \mid y \in x.vs, z \in x.vs, y_1 \in y.v_1, z_1 \in z.v_2\}](X, Y)\\ \equiv\ \ &\mathbf{MR}\ [\lambda x.\{\langle k : \pi_\kappa(x), v : \theta(x, z)\rangle \mid z \in Y, \pi_\iota(z) \approx \pi_\kappa(x)\}][\lambda x.\mathbf{id}_R](X).\end{aligned}$$

The θ is a convenience notation for merging the data from two tuples into a single tuple. It is easily NRC expressible as long as we know the the tuple types and how to deal with field name collisions.

As was shown in previous paragraph *JOIN*'s result is usually materialized in the reducer, but if one dataset is small enough it could be materialized in the mapper. We refer to the first as the reduce-side *JOIN*, and to the second one as the map-side *JOIN*. In the following we are discussing combining multiple *JOIN*s together, assuming they join on the same key, and we have to consider three cases: combining two map-side *JOIN*s, combining a map-side *JOIN* with a reduce-side *JOIN* and combining two reduce-side *JOIN*s. Note that any computation in the mapper, before creating the intermediate key, can be seen as a preprocessing. Thus combining a map-side *JOIN* with any other join can be treated as adding an additional preprocessing before the actual MR routine. Hence the first two cases are easy.

The last case, namely combining reduce-side *JOIN*s, and also combining *JOIN*s with *GROUP BY*s on the same key, are generalized by combining *COGROUP* operators which we present here. We define a family of rewrite rules, depending on the number of inputs, and show only an example for three inputs, as generalization is simple:

$$\begin{aligned}
&\mathbf{MR}\ [\lambda x.\{\langle k : x.k, v : \chi^2_1(x.v)\rangle\rangle\}, \lambda x.\{\langle k : \pi_\iota(x), v : \chi^2_2(x)\rangle\}]\\
&\quad [\lambda x.\langle k : x.k, v : \langle\{z \mid y \in x.vs, z \in y.v_1.1\},\\
&\quad \{z \mid y \in x.vs, z \in y.v_1.2\}, \{z \mid y \in x.vs, z \in y.v_2\}\rangle\rangle](\mathit{inner(X,Y)}, Z)\\
\equiv\ &\mathbf{MR}\ [\lambda x.\{\langle k : \pi_\kappa(x), v : \chi^3_1(x)\rangle\}, \lambda x.\{\langle k : \pi_\iota(x), v : \chi^3_2(x)\rangle\},\\
&\quad \lambda x.\{\langle k : \pi_\zeta(x), v : \chi^3_3(x)\rangle\}][\lambda x.\langle x.k, \{z \mid y \in x.vs, z \in y.v_1\},\\
&\quad \{z \mid y \in x.vs, z \in y.v_2\}, \{z \mid z \in x.vs, z \in y.v_3\}\rangle](X, Y, Z),
\end{aligned}$$

where *inner* is a *COGROUP*:

$$\begin{aligned}
&\mathbf{MR}\ [\lambda x.\{\langle k : \pi_\kappa(x), v : \chi^2_1(x)\rangle\}, \lambda x.\{\langle k : \pi_\iota(x), v : \chi^2_2(x)\rangle\}]\\
&\quad [\lambda x.\langle k : x.k, v : \langle\{z \mid y \in x.vs, z \in y.v_1\}, \{z \mid y \in x.vs, z \in y.v_2\}\rangle\rangle](X, Y).
\end{aligned}$$

Fusing *cmaps* (5) is a plain NRC rewrite rule, as it is roughly the same as fusing the comprehensions:

$$\begin{aligned}
\mathit{cmap}(f)(\mathit{cmap}(g)D) &= \{y \mid x \in \{g(x) \mid x \in D\}, y \in f(x)\}\\
&\equiv \mathit{cmap}(\lambda x.\{z \mid y \in g(x), z \in f(y)\})(D).
\end{aligned}$$

Note that a composition of *cmap*s is also a *cmap*, hence there is actually never a need to more than a single *cmap* between other operators.

We deal with the last two items (6) and (7) together, as projections and filterings are just a special case of the *cmap* operator. There are two cases of possible fusions. Either the *cmap* may be done on an input of *COGROUP* or *JOIN*, or on their output. Both those cases can be easily represented as rewrite rules. We denote the *cmap*s UDF by *f* and present only the right hands of the rules, as the left hand sides are straightforward pipelinings of MR routines corresponding respectively to the given constructs for *cmap* before and after *COGROUP*:

$$\begin{aligned}
&\ldots = \mathbf{MR}\ [\lambda x.\{\langle k : \pi_\kappa(x), v : \chi^2_1(f(x))\rangle\}, \lambda x.\{\langle k : \pi_\iota(x), v : \chi^2_2(x)\rangle\}]\\
&\quad [\lambda x.\langle k : x.k, v : \langle\{z \mid y \in x.vs, z \in y.v_1\}, \{z \mid y \in x.vs, z \in y.v_2\}\rangle\rangle](X, Y),\\
&\ldots = \mathbf{MR}\ [\lambda x.\{\langle k : \pi_\kappa(x), v : \chi^2_1(x)\rangle\}, \lambda x.\{\langle k : \pi_\iota(x), v : \chi^2_2(x)\rangle\}]\\
&\quad [\lambda x.f(\langle k : x.k, v : \langle\{z \mid y \in x.vs, z \in y.v_1\}, \{z \mid y \in x.vs, z \in y.v_2\}\rangle\rangle)](X, Y).
\end{aligned}$$

8 Conclusion

In this paper we have demonstrated that the Nested Relational Calculus is a suitable language to formulate and reason about MR programs for nested data. It is declarative and higher level than MR, but in some ways lower level than MRQL thus allowing a bit more precise refined optimizations. We showed that MR programs can be expressed in NRC when allowed the same class of UDFs. We also showed that the NRC formalism can express all constructs and optimizations found in Hive, Pig Latin and MRQL. Moreover, NRC is suitable both for writing high-level queries and transformations, as well as MR-based physical evaluation plans when extended with the appropriate constructs. This has the benefit of allowing optimization through rewriting essentially the same formalism, which is not the case for any of the former higher-level MR languages.

Our framework allows for a clear representation of MR programs, which is essential for reasoning about particular programs or the framework in general. NRC is a well defined and thoroughly described language, which has the appropriate level of abstraction to specify the class of MR algorithms we want to concentrate on. It is important that this language is well-designed, much smaller and with a much simpler semantics than other languages than were used to describe MR, like Java or Haskell. This is the reason we think that our work can be potentially more effective than [5,4].

The higher-level goal of this research is to build a query optimization module that takes as input an NRC expression and translates it into an efficient MR program that can be executed on a MapReduce backend. In future work we will therefore investigate further to what extent NRC and NRC-MR allow for meaningful optimizations through rewriting, either heuristically or cost-based. Moreover, we will investigate the problem of deciding which sub-expressions can be usefully mapped to an MR step, and how this mapping should look in order to obtain an efficient query evaluation plan. This will involve investigating which cost-models are effective for the different types of MapReduce backends.

References

1. Afrati, F.N., Sarma, A.D., Salihoglu, S., Ullman, J.D.: Vision paper: Towards an understanding of the limits of map-reduce computation. CoRR, abs/1204.1754 (2012)
2. Borthakur, D., Gray, J., Sarma, J.S., Muthukkaruppan, K., Spiegelberg, N., Kuang, H., Ranganathan, K., Molkov, D., Menon, A., Rash, S., Schmidt, R., Aiyer, A.: Apache hadoop goes realtime at facebook. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, pp. 1071–1080. ACM, New York (2011)
3. Buneman, P., Naqvi, S.A., Tannen, V., Wong, L.: Principles of programming with complex objects and collection types. Theor. Comput. Sci. 149(1), 3–48 (1995)
4. Cafarella, M.J., Ré, C.: Manimal: relational optimization for data-intensive programs. In: Procceedings of the 13th International Workshop on the Web and Databases, WebDB 2010, pp. 10:1–10:6. ACM, New York (2010)

5. Chambers, C., Raniwala, A., Perry, F., Adams, S., Henry, R.R., Bradshaw, R., Weizenbaum, N.: Flumejava: easy, efficient data-parallel pipelines. In: Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2010, pp. 363–375. ACM, New York (2010)
6. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: OSDI, pp. 137–150 (2004)
7. Van den Bussche, J., Vansummeren, S.: Polymorphic type inference for the named nested relational calculus. ACM Trans. Comput. Log. 9(1) (2007)
8. Fegaras, L., Li, C., Gupta, U.: An optimization framework for map-reduce queries. In: EDBT, pp. 26–37 (2012)
9. Fegaras, L., Maier, D.: Optimizing object queries using an effective calculus. ACM Trans. Database Syst. 25(4), 457–516 (2000)
10. Gates, A., Natkovich, O., Chopra, S., Kamath, P., Narayanam, S., Olston, C., Reed, B., Srinivasan, S., Srivastava, U.: Building a highlevel dataflow system on top of mapreduce: The pig experience. PVLDB 2(2), 1414–1425 (2009)
11. Ghemawat, S., Gobioff, H., Leung, S.-T.: The google file system. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, SOSP 2003, pp. 29–43. ACM, New York (2003)
12. Karloff, H.J., Suri, S., Vassilvitskii, S.: A model of computation for mapreduce. In: SODA, pp. 938–948 (2010)
13. Lämmel, R.: Google's MapReduce Programming Model – Revisited. Science of Computer Programming 70(1), 1–30 (2008)
14. Malewicz, G., Austern, M.H., Bik, A.J.C., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: a system for large-scale graph processing. SPAA 48 (2009)
15. Pike, R., Dorward, S., Griesemer, R., Quinlan, S.: Interpreting the data: Parallel analysis with sawzall. Scientific Programming 13(4), 277–298 (2005)
16. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive - a warehousing solution over a map-reduce framework. PVLDB 2(2), 1626–1629 (2009)
17. Van den Bussche, J., Van Gucht, D., Vansummeren, S.: A crash course on database queries. In: Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2007, pp. 143–154. ACM, New York (2007)
18. Wadler, P.: Comprehending monads. Mathematical Structures in Computer Science 2(4), 461–493 (1992)

Bisimulation Reduction of Big Graphs on MapReduce

Yongming Luo[1], Yannick de Lange[1], George H.L. Fletcher[1],
Paul De Bra[1], Jan Hidders[2], and Yuqing Wu[3]

[1] Eindhoven University of Technology, The Netherlands
{y.luo@,y.m.d.lange@student.,g.h.l.fletcher@,P.M.E.d.Bra@}tue.nl
[2] Delft University of Technology, The Netherlands
a.j.h.hidders@tudelft.nl
[3] Indiana University, Bloomington, USA
yuqwu@cs.indiana.edu

Abstract. Computing the bisimulation partition of a graph is a fundamental problem which plays a key role in a wide range of basic applications. Intuitively, two nodes in a graph are bisimilar if they share basic structural properties such as labeling and neighborhood topology. In data management, reducing a graph under bisimulation equivalence is a crucial step, e.g., for indexing the graph for efficient query processing. Often, graphs of interest in the real world are massive; examples include social networks and linked open data. For analytics on such graphs, it is becoming increasingly infeasible to rely on in-memory or even I/O-efficient solutions. Hence, a trend in Big Data analytics is the use of distributed computing frameworks such as MapReduce. While there are both internal and external memory solutions for efficiently computing bisimulation, there is, to our knowledge, no effective MapReduce-based solution for bisimulation. Motivated by these observations we propose in this paper the first efficient MapReduce-based algorithm for computing the bisimulation partition of massive graphs. We also detail several optimizations for handling the data skew which often arises in real-world graphs. The results of an extensive empirical study are presented which demonstrate the effectiveness and scalability of our solution.

1 Introduction

Recently, graph analytics has drawn increasing attention from the data management, semantic web, and many other research communities. Graphs of interest, such as social networks, the web graph, and linked open data, are typically on the order of billions of nodes and edges. In such cases, single-machine in-memory solutions for reasoning over graphs are often infeasible. Naturally, research has turned to external memory and distributed solutions for graph reasoning. External memory algorithms often suffice, but their performance typically scales (almost) linearly with graph size (usually the number of graph edges), which is then limited by the throughput of the I/O devices attached to the system. In this respect, distributed and parallel algorithms become attractive. Ideally, a

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 189–203, 2013.

well-designed distributed algorithm would scale (roughly) linearly with the size of the computing resources it has, making use of the parallelism of the infrastructure. Though there are many alternatives, recently the MapReduce platform [8] has become a *de-facto* parallel processing platform for reasoning over Big Data such as real-world graphs, with wide adoption in both industry and research.

Among fundamental graph problems, the bisimulation partition problem plays a key role in a surprising range of basic applications [24]. Informally, the bisimulation partition of a graph is an assignment of each node n of the graph to a unique block consisting of all nodes having the same structural properties as n (e.g., node label and neighborhood topology). In data management, variations of bisimulation play a fundamental role in constructing structural indexes for XML and RDF databases [21,23], and many other applications for general graph data such as compression [4,11], query processing [16], and data analytics [10]. Being well studied for decades, many main-memory efficient algorithms have been developed for bisimulation partitioning (e.g., [9,22]). The state-of-the-art I/O efficient algorithm takes just under one day to compute a standard "localized" variant of bisimulation on a graph with 1.4 billion edges on commodity hardware [20]. This cost can be a potential bottleneck since bisimulation partitioning is typically one step in a larger workflow (e.g., preparing the graph for indexing and query processing).

Contributions. Motivated by these observations, we have studied the effective use of the MapReduce framework for accelerating the computation of bisimulation partitions of massive graphs. In this paper we present, to our knowledge, the first efficient MapReduce-based algorithm for localized bisimulation partitioning. We further present strategies for dealing with various types of skew which occur in real-world graphs. We discuss the results of extensive experiments which show that our approach is effective and scalable, being up to an order of magnitude faster than the state of the art. As a prototypical graph problem, we hope that sharing our experiences with graph bisimulation will stimulate further progress in the community on MapReduce-based solutions for reasoning and analytics over massive graphs.

Related Work. While there has been work on distributed computation of bisimulation partitions, existing approaches (e.g., [3]) are not developed for the MapReduce platform, and hence are not directly applicable to our problem. Research has been conducted to investigate using the MapReduce framework to solve graph problems [6,18] right after the framework was proposed. A major issue here is dealing with data skew in graphs. Indeed, skew is ubiquitous in real-world graphs [5]. During our investigation, we also experienced various types of skew from graph bisimulation, as we discuss below. There has been much progress done from the system side to tackle this problem. The main approach in this literature is to devise mechanisms to estimate costs of MapReduce systems (e.g., [13]) and modify the system to mitigate the skew effects, both statically [12,15,17] and dynamically [17,25], so that the modification is transparent to the users. However, it is still possible to gain much efficiency by dealing with skew from the algorithm design perspective [19], as we do in the novel work we present in this paper.

Paper Organization. In the next section we give a brief description of localized bisimulation and MapReduce. We then describe in Section 3 our base algorithm for computing localized bisimulation partitions using MapReduce. Next, Section 4 presents optimizations of the base algorithm, to deal with the common problem of skewed data. Section 5 presents the results of our empirical study. We then conclude in Section 6 with a discussion of future directions for research.

2 Preliminaries

2.1 Localized Bisimulation Partitions

Our data model is that of finite directed node- and edge-labeled graphs $\langle N, E, \lambda_N, \lambda_E \rangle$, where N is a finite set of nodes, $E \subseteq N \times N$ is a set of edges, λ_N is a function from N to a set of node labels $\mathcal{L}_N$, and λ_E is a function from E to a set of edge labels $\mathcal{L}_E$.

The localized bisimulation partition of graph $G = \langle N, E, \lambda_N, \lambda_E \rangle$ is based on the *k-bisimilar* equivalence relation.

Definition 1. *Let $G = \langle N, E, \lambda_N, \lambda_E \rangle$ be a graph and $k \geq 0$. Nodes $u, v \in N$ are called k-*bisimilar *(denoted as $u \approx^k v$), iff the following holds:*

1. *$\lambda_N(u) = \lambda_N(v)$,*
2. *if $k > 0$, then for any edge $(u, u') \in E$, there exists an edge $(v, v') \in E$, such that $u' \approx^{k-1} v'$ and $\lambda_E(u, u') = \lambda_E(v, v')$, and*
3. *if $k > 0$, then for any edge $(v, v') \in E$, there exists an edge $(u, u') \in E$, such that $v' \approx^{k-1} u'$ and $\lambda_E(v, v') = \lambda_E(u, u')$.*

Given the k-bisimulation relation on a graph G, we can assign a unique partition identifier (e.g., an integer) to each set of k-bisimilar nodes in G. For node $u \in N$ and relation $\approx^k$, we write $pId_k(u)$ to denote u's k-partition identifier, and we call pId_k a *k-partition identifier function.*

Definition 2. *Let $G = \langle N, E, \lambda_N, \lambda_E \rangle$ be a graph, $k \geq 0$, and $\{pId_0, \ldots, pId_k\}$ be a set of i-partition identifier functions for G, for $0 \leq i \leq k$. The k* bisimulation signature *of node $u \in N$ is the pair $sig_k(u) = (pId_0(u), L)$ where:*

$$L = \begin{cases} \emptyset & \text{if } k = 0, \\ \{(\lambda_E(u, u'), pId_{k-1}(u')) \mid (u, u') \in E\} & \text{if } k > 0. \end{cases}$$

We then have the following fact.

Proposition 1 ([20]). *$pId_k(u) = pId_k(v)$ iff $sig_k(u) = sig_k(v), k \geq 0$.*

Since there is a one-to-one mapping between a node's signature and its partition identifier, we can construct $sig_k(u)$ ($\forall u \in N, k \geq 0$), assign $pId_k(u)$ according to $sig_k(u)$, and then use $pId_k(u)$ to construct $sig_{k+1}(u)$, and so on. We call each such construct-assign computing cycle an *iteration.* This signature-based approach is robust, with effective application in non-MapReduce environments (e.g., [3,20]). We refer for a more detailed discussion of localized bisimulation to Luo et al. [20].

Example. Consider the graph in Figure 1. In iteration 0, nodes are partitioned into blocks *P1* and *P2* (indicated by different colors), based on the node label A and B (Def. 1). Then in iteration 1, from Def. 2, we have $sig_1(\mathit{1}) = (1, \{(w, P1), (l, P2)\})$ and $sig_1(\mathit{2}) = (1, \{(w, P1), (l, P2)\})$, which indicates that $pId_1(\mathit{1}) = pId_1(\mathit{2})$ (Prop. 1). If we further compute 2-bisimulation, we see that $sig_2(\mathit{1}) \neq sig_2(\mathit{2})$, and we conclude that nodes 1 and 2 are not 2-bisimilar, and block *P1* will split.

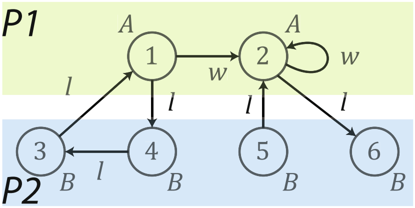


Fig. 1. Example graph

The partition blocks and their relations (i.e., a “structural index”) can be seen as an abstraction of the real graph, to be used, for example, to filter unnecessary graph matching during query processing [11,23]. A larger k leads to a more refined partition, which results in a larger structural index. So there is a trade-off between k and the space we have for holding the structural index. In practice, though, we see that a small k value (e.g., $k \leq 5$) is already sufficient for query processing. In our analysis below, we compute the k-bisimulation result up to $k = 10$, which is enough to show all the behaviors of interest for structural indexes.

2.2 MapReduce Framework

The MapReduce programming model [8] is designed to process large datasets in parallel. A MapReduce *job* takes a set of key/value pairs as input and outputs another set of key/value pairs. A MapReduce *program* consists of a series of MapReduce jobs, where each MapReduce job implements a *map* and a *reduce* function (“[ ]” means a list of elements):

$$\begin{array}{lll} \mathit{map} & (k_1, v_1) & \rightarrow [(k_2, v_2)] \\ \mathit{reduce} & (k_2, [v_2]) & \rightarrow [(k_3, v_3)]. \end{array}$$

The *map* function takes key/value pair (k_1, v_1) as the input, emits a list of key/value pairs (k_2, v_2). In the *reduce* function, all values with the same key are grouped together as a list of values v_2 and are processed to emit another list of key/value pairs (k_3, v_3). Users define the *map* and *reduce* functions, letting the framework take care of all other aspects of the computation (synchronization, I/O, fault tolerance, etc.).

The open source Hadoop implementation of the MapReduce framework is considered to have production quality and is widely used in industry and research [14]. Hadoop is often used together with the Hadoop Distributed File System (HDFS), which is designed to provide high-throughput access to application data. Besides *map* and *reduce* functions, in Hadoop a user can also write a custom *partition* function, which is applied after the *map* function to specify to which reducer each key/value pair should go. In our work we use Hadoop for validating our solutions, making use of the *partition* function as well.

3 Bisimulation Partitioning with MapReduce

For a graph $G = \langle N, E, \lambda_N, \lambda_E \rangle$, we arbitrarily assign unique (integer) identifiers to each node of N. Our algorithm for computing the k-bisimulation partition of G has two input tables: a node table (denoted as N_t) and an edge table (denoted as E_t). Both tables are plain files of sequential records of nodes and edges of G, resp., stored on HDFS. The schema of table N_t is as follows:

nId	node identifier
pId_{0_nId}	0 *bisimulation* partition identifier for the given *nId*
pId_{new_nId}	*bisimulation* partition identifier for the given *nId* from the current computation iteration

The schema of table E_t is as follows:

sId	source node identifier
tId	target node identifier
eLabel	edge label
pId_{old_tId}	*bisimulation* partition identifier for the given *tId* from the last computation iteration

By combining the idea of Proposition 1 and the MapReduce programming model, we can sketch an algorithm for k-bisimulation using MapReduce, with the following workflow: for each iteration i $(0 \leq i \leq k)$, we first construct the signatures for all nodes, then assign partition identifiers for all unique signatures, and pass the information to the next iteration. In our algorithm, each iteration then consists of three MapReduce tasks:

1. task `Signature` performs a merge join of N_t and E_t, and create signatures;
2. task `Identifier` distributes signatures to reducers and assigns partition identifiers; and,
3. task `RePartition` sorts N_t to prepare it for the next iteration.

Note that a preprocessing step is executed to guarantee the correctness of the map-side join in task `Signature`. We next explain each task in details.

3.1 Task `Signature`

Task `Signature` (Algorithm 1) first performs a sort merge join of N_t and E_t, filling in the pId_{old_tId} column of E_t with pId_{new_nId} of N_t. This is achieved in the map function using a map-side join [18]. Then records are emitted grouping by *nId* in N_t and *sId* in E_t. In the reduce function, all information to construct a signature resides in [*value*]. So the major part of the function is to iterate through [*value*] and construct the signature according to Definition 2. After doing so, the node identifier along with its pId_{0_nId} value and signature are emitted to the next task. Note that the key/value pair input of `SignatureMapper` indicates the fragments of N_t and E_t that need to be joined. The fragments need to be preprocessed before the algorithm runs.

Algorithm 1. task `Signature`

```
procedure SignatureMapper($key$, $value$)
    perform map-side equi-join of $N_t$ and $E_t$ on $nId$ and $tId$, fill in $pId_{old\_tId}$ with $pId_{new\_nId}$, put all rows of $N_t$ and $E_t$ into $records$
    for $row$ in $records$ do
        if $row$ is from $E_t$ then
            emit ($sId$, the rest of the $row$)
        else if $row$ is from $N_t$ then
            emit ($nId$, the rest of the $row$)

procedure SignatureReducer($key$, [$value$])                ▷ key is $nId$ or $sId$
    $pairset \leftarrow \{\}$
    for $value$ in [$value$] do
        if ($key$,$value$) is from $N_t$ then
            $pId_{0\_nId} \leftarrow value.pId_{0\_nId}$                ▷ record $pId_{0\_nId}$
        else if ($key$,$value$) is from $E_t$ then
            $pairset \leftarrow pairset \cup \{(eLabel, pId_{old\_tId})\}$
    sort elements in $pairset$ lexicographically, first on $eLabel$ then on $pId_{old\_tId}$
    $signature \leftarrow (pId_{0\_nId}, pairset)$
    emit ($key$, ($pId_{0\_nId}$, $signature$))
```

3.2 Task `Identifier`

On a single machine, in order to assign distinct values for signatures, we only need a dictionary-like data structure. In a distributed environment, on the other hand, some extra work has to be done. The `Identifier` task (Algorithm 2) is designed for this purpose. The map function distributes nodes of the same signature to the same reducer, so that in the reduce function, each reducer only needs to check locally whether the given signature is assigned an identifier or not; if not, then a new identifier is generated and assigned to the signature. To assign identifiers without collisions across reducers, we specify a non-overlapping identifier range each reducer can use beforehand. For instance, reducer i can generate identifiers in the range of $[i \times |N|, (i+1) \times |N|)$.

Algorithm 2. task `Identifier`

```
procedure IdentifierMapper($nId$, ($pId_{0\_nId}$, $signature$))
    emit ($signature$, ($nId$,$pId_{0\_nId}$))

procedure IdentifierReducer($signature$, [($nId$, $pId_{0\_nId}$)])
    $pId_{new\_nId} \leftarrow$ get unique identifier for $signature$
    for ($nId$, $pId_{0\_nId}$) in [($nId$, $pId_{0\_nId}$)] do
        emit ($nId$, ($pId_{0\_nId}$, $pId_{new\_nId}$))
```

3.3 Task `RePartition`

The output of task `Identifier` is N_t filled with partition identifiers from the current iteration, but consists of file fragments partitioned by signature. In order to perform a map-side join with E_t in task Signature in the next iteration, N_t needs to be sorted and partitioned on *nId*, which is the main job of task RePartition (Algorithm 3). This task makes use of the MapReduce framework to do the sorting and grouping.

Algorithm 3. task RePartition

```
procedure RePartitionMapper($nId$, $(pId_{0\_nId}, pId_{new\_nId})$)
    emit ($nId$, $(pId_{0\_nId}, pId_{new\_nId})$)                ▷ do nothing

procedure RePartitionReducer($nId$, $[(pId_{0\_nId}, pId_{new\_nId})]$)
    for $(pId_{0\_nId}, pId_{new\_nId})$ in $[(pId_{0\_nId}, pId_{new\_nId})]$ do
        emit ($nId$, $(pId_{0\_nId}, pId_{new\_nId})$)
```

3.4 Running Example

We illustrate our algorithm on the example graph of Figure 1. In Figures 2(a), 2(b) and 2(c), we show the input and output of the map and reduce phases of tasks Signature, Identifier and RePartition, respectively, for the first iteration ($i = 0$), with two reducers (bounded with gray boxes) in use.

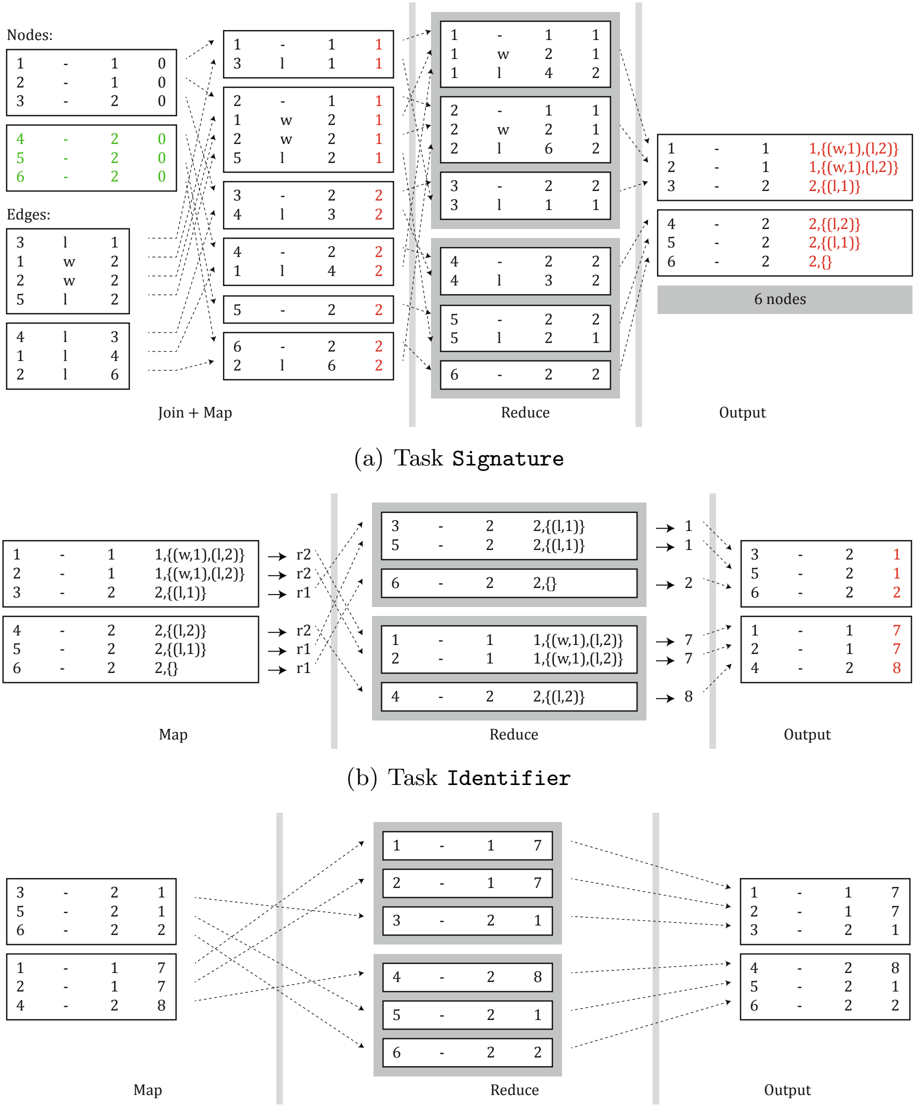
4 Strategies for Dealing with Skew in Graphs

4.1 Data and Skew

In our investigations, we used a variety of graph datasets to validate the results of our algorithm, namely: Twitter (41.65M, 1468.37M), Jamendo (0.48M, 1.05M), LinkedMDB (2.33M, 6.15M), DBLP (23M, 50.2M), WikiLinks (5.71M, 130.16M), DBPedia (38.62M, 115.3M), Power (8.39M, 200M), and Random (10M, 200M); the numbers in the parenthesis indicates the node count and edge count of the graph, resp.. Among these, Jamendo, LinkedMDB, DBLP, WikiLinks, DBPedia, and Twitter are real-world graphs described further in Luo et al. [20]. Random and Power are synthetic graphs generated using GTgraph [2], where Random is generated by adding edges between nodes randomly, and Power is generated following the power-law degree distribution and small-world property.

During investigation of our base algorithm (further details below in Section 5), we witnessed a highly skewed workload among mappers and reducers. Figure 4(a) illustrates this on the various datasets, showing the running time for different reducers for the three tasks in the algorithm. Each spike is a time measurement for one reducer. The running time in each task is sorted in a descending order. We see that indeed some reducers carry a significantly disproportionate workload. This skew slows down the whole process since the task is only complete when the slowest worker finishes.

From this empirical observation, we trace back the behavior to the data. Indeed, the partition result is skewed in many ways. For example, in Figure 3, we show the cumulative distribution of partition block size, i.e., number of nodes



(a) Task `Signature`

(b) Task `Identifier`

(c) Task `RePartition`

Fig. 2. Input and output of the three tasks of our algorithm for the example graph of Figure 1 (iteration $i = 0$)

assigned to the block, to the number of partition blocks having the given size, for the real-world graphs described above. We see that for all of the datasets, block size shows a power-law distribution property.

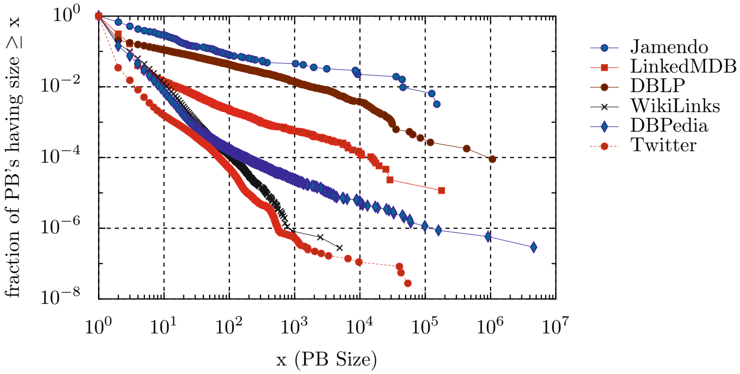


Fig. 3. Cumulative distribution of partition block size (PB Size) to partition blocks having the given size for real-world graphs

This indicates the need to rethink the design of our base algorithm. Recall that at the end of the map function of Algorithm 2, nodes are distributed to reducers based on their signatures. Since some of the signatures are associated with many nodes (as we see in Figure 3), the workload is inevitably unbalanced. This explains the reducers' behavior in Figure 4(a) as well. In the following, we propose several strategies to handle such circumstances.

4.2 Strategy 1: Introduce another Task `Merge`

Recall from Section 3.2 that nodes with the same signature must be distributed to the same reducer, for otherwise the assignment of partition block identifiers cannot be guaranteed to be correct. This could be relaxed, however, if we introduce another task, which we call `Merge`. Suppose that when we output the partition result (without the guarantee that nodes with the same signature go to the same reducer), for each reducer, we also output a mapping of signatures to partition identifiers. Then in task `Merge`, we could merge the partition IDs based on the signature value, and output a local_pid to global_pid mapping. Then in the task `RePartition`, another map-side join is introduced to replace the local_pid with the global_pid. Because the signature itself is not correlated with the partition block size, the skew on partition block size should be eliminated. We discuss the implementation details in the following.

In N_t assume we have an additional field holding the partition size of the node from the previous iteration, named $pSize_{old}$, and let $MR_{Load} = \frac{|N|}{number\ of\ reducers}$. We define rand(x) to return a random integer from 0 to x, and % as the modulo operator.

We first change the partition function for `Identifier` (Algorithm 4). In this case, for nodes whose associated $pSize_{old}$ are above the threshold, we do not guarantee that they end up in the same reducer, but make sure that they

Algorithm 4. modified partition function for task `Identifier`

```
procedure IDENTIFIER_GETPARTITION                      ▷ for each key/value pair
    if $pSize_{old} > threshold$ then
        $n = pSize_{old}/MR_{Load}$                       ▷ numbers of reducers we need
        return ($signature$.hashcode()+rand($n$)) % $number\ of\ reducers$
    else
        return $signature$.hashcode() % $number\ of\ reducers$
```

are distributed to at most n reducers. Then we come to the reduce phase for `Identifier` (Algorithm 5). Here we also output the local partition size (named $pSize_{new}$) for each node.

Then the task `Merge` (Algorithm 6) will create the mapping between the locally assigned pId_{new_nId} and *global_pid*.

Algorithm 5. modified reduce function for task `Identifier`

```
procedure IDENTIFIERREDUCER($signature$, [($nId$,$pId_{0\_nId}$,$pSize_{old}$)])
    $pId_{new\_nId} \leftarrow$ get unique identifier for $signature$
    $pSize_{new} \leftarrow$ size of [($nId$,$pId_{0\_nId}$, $pSize_{old}$)]
    for ($nId$, $pId_{0\_nId}$, $pSize_{old}$) in [($nId$, $pId_{0\_nId}$, $pSize_{old}$)] do
        emit ($nId$, ($pId_{0\_nId}$, $pId_{new\_nId}$))
        emit ($signature$, ($pId_{new\_nId}$, $pSize_{new}$))
```

Algorithm 6. task `Merge`

```
procedure MERGEMAPPER($signature$, ($pId_{new\_nId}$, $pSize_{new}$))
    emit ($signature$, ($pId_{new\_nId}$, $pSize_{new}$))                 ▷ do nothing

procedure MERGEREDUCER($signature$, [($pId_{new\_nId}$, $pSize_{new}$)])
    $global\_pid\_count \leftarrow 0$
    $global\_pid \leftarrow$ get unique identifier for $signature$
    for ($pId_{new\_nId}$, $pSize_{new}$) in [($pId_{new\_nId}$, $pSize_{new}$)] do
        $global\_pid\_count \leftarrow global\_pid\_count + pSize_{new}$
        emit ($global\_pid$, ($pId_{new\_nId}$))          ▷ the local - global mapping
    emit ($global\_pid$, $global\_pid\_count$)
```

Finally at the beginning of task `RePartition`, the partition identifiers are unified by a merge join of the local_pid - global_pid mapping table and N_t. We achieve this by distributing the local_pid - global_pid table to all mappers before the task begins, with the help of Hadoop's distributed cache. Also, global_pid_count is updated in N_t.

While this is a general solution for dealing with skew, the extra `Merge` task introduces additional overhead. In the case of heavy skew, some signatures will produce large map files and performing merging might become a bottleneck. This indicates the need for a more specialized solution to deal with heavy skew.

4.3 Strategy 2: Top-K Signature-Partition Identifier Mapping

One observation of Figure 3 is that only a few partition blocks are heavily skewed. To handle these outliers, at the end of task `Signature`, besides emitting N_t, we can also emit, for each reducer, an aggregation count of signature appearances. Then a merge is performed among all the counts, to identify the most frequent K signatures and fix signature-partition identifier mappings for these popular partition blocks. This mapping is small enough to be distributed to all cluster nodes as a global variable by Hadoop, so that when dealing with these signatures, processing time becomes constant. As a result, in task `Identifier`, nodes with such signatures can be distributed randomly across reducers.

There are certain drawbacks to this method. First, the output top-K frequent signatures are aggregated from local top-K frequent values (with respect to each reducer), but globally we only use these values as an estimation of the real counts. Second, the step where signature counts have to be output and merged becomes a bottleneck of the whole workflow. Last but not least, users have to specify K before processing, which may be either too high or too low.

However, in the case of extreme skew on the partition block sizes, for most of the real world datasets, there are only a few partition blocks which delay the whole process, even for very large datasets. So when we adopt this strategy, we can choose a very small K value and still achieve good results, without introducing another MapReduce task. This is validated in Section 5.1.

5 Empirical Analysis

Our experiments are executed on the Hadoop cluster at SURFsara in Amsterdam, The Netherlands.[1] This cluster consists of 72 Nodes (66 DataNodes & TaskTrackers and 6 HeadNodes), with each node equipped with 4 x 2TB HDD, 8 core CPU 2.0 GHz and 64GB RAM. In total, there are 528 map and 528 reduce processes, and 460 TB HDFS space. The cluster is running Cloudera CDH3 distribution with Apache Hadoop 0.20.205. All algorithms are written in Java 1.6. The datasets we use are described in Section 4.1. A more detailed description of the empirical study can be found in de Lange [7].

5.1 Impact on Workload Skew of the Top-K Strategy

Figure 4(b) shows the cluster workload after we create a identifier mapping for the top-2 signature-partitions from Section 4.3. We see that, when compared with Figure 4(a), the skew in running time per reducer is eliminated by the strategy. This means that workload is better distributed, thus lowering the running time per iteration and, in turn, the whole algorithm.

[1] `https://www.surfsara.nl`

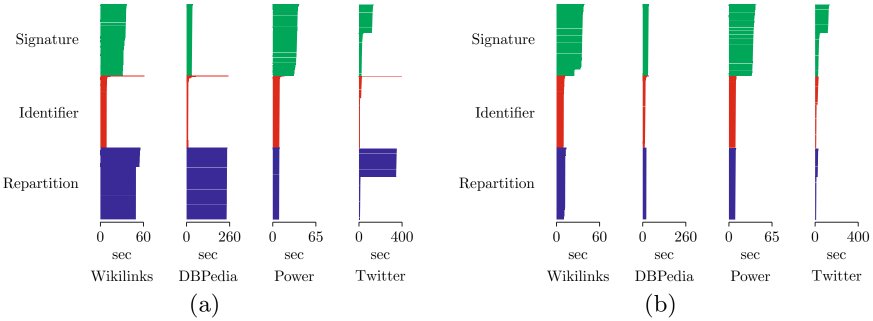


Fig. 4. Workload skewness for base algorithm (a) and for top-2 signature strategy (b)

5.2 Overall Performance Comparison

In Figure 5, we present an overall performance comparison for computing 10-bisimulation on each graph with: our base algorithm (Base), the merge (Merge) and top-K (Top-K Signature) skew strategies, and the state of the art single-machine external memory algorithm (I/O Efficient) [20]. For the Top-K Signature strategy, we set $K = 2$ which, from our observation in Section 5.1, gives excellent skew reduction with low additional cost. For the Merge optimization we used a threshold of $1 \times MR_{load}$ such that each partition block larger than the optimal reducer block size is distributed among additional reducers. Furthermore, for each dataset, we choose $3 \times \lceil \frac{edge\ table\ size}{64\ MB} \rceil$ number of reducers, which has been tested to lead to the minimum elapsed time. We empirically observed that increasing the number of reducers beyond this does not improve performance. Indeed, adding further maps and reducers at this point negatively impacts the running time due to framework overhead. Each experiment was repeated five times, and the data point reported is the average of the middle three measurements.

We immediately observe from Figure 5 that for all datasets, among the Base algorithm and its two optimizations, there are always at least two solutions

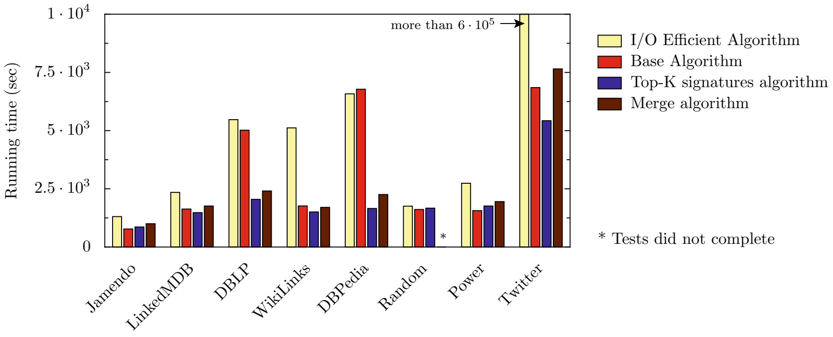


Fig. 5. Overall performance comparison

which perform better than the I/O efficient solution. For small datasets such as Jamendo and LinkedMDB, this is obvious, since in these cases only 1 or 2 reducers are used, so that the algorithm turns into a single-machine in-memory algorithm. When the size of the datasets increases, the value of MapReduce becomes more visible, with up to an order of magnitude improvement in the running time for the MapReduce-based solutions. We also observe that the skew amelioration strategies give excellent overall performance on these big graphs, with 2 or 3 times of improvement over our Base algorithm in the case of the highly skewed graphs such as DBLP and DBPedia. Finally, we observe that, relative to the top-K strategy, the merge skew-strategy is mostly placed at a disadvantage due to its inherent overhead costs.

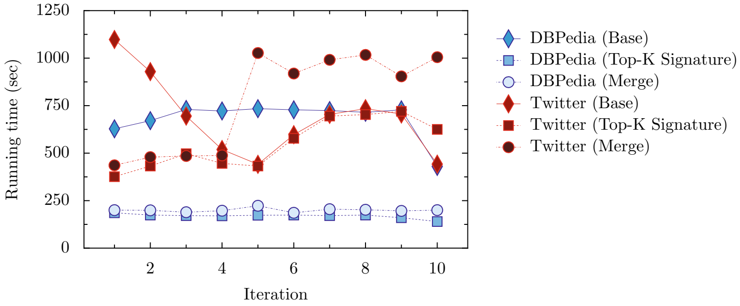


Fig. 6. Performance comparison per iteration for Twitter and DBPedia

To further study the different behaviors among the MapReduce-based solutions, we plot the running time per iteration of the three solutions for DBPedia and Twitter in Figure 6. We see that for the Twitter dataset, in the first four iterations the skew is severe for the Base algorithm, while the two optimization strategies handle it well. After iteration 5, the overhead of the Merge strategy becomes non-negligible, which is due to the bigger mapping files the `Identifier` produces. For the DBPedia dataset, on the other hand, the two strategies perform consistently better than the Base algorithm.

Over all, based on our experiments, we note that our algorithm's performance is stable, i.e., essentially constant in running time as the number of maps and reducers is scaled with the input graph size.

6 Concluding Remarks

In this paper, we have presented, to our knowledge, the first general-purpose algorithm for effectively computing localized bisimulation partitions of big graphs using the MapReduce programming model. We witnessed a skewed workload during algorithm execution, and proposed two strategies to eliminate such skew from an algorithm design perspective. An extensive empirical study confirmed that our algorithm not only efficiently produces the bisimulation partition result, but also scales well with the MapReduce infrastructure, with an order of magnitude performance improvement over the state of the art on real-world graphs.

We close by indicating several interesting avenues for further investigation. First, there are additional basic sources of skew which may impact performance of our algorithm, such as skew on signature sizes and skew on the structure of the bisimulation partition itself. Therefore, further optimizations should be investigated to handle these additional forms of skew. Second, in Section 5.2 we see that all three proposed solutions have their best performance for some dataset, therefore it would be interesting to study the cost model of the MapReduce framework and combine the information (e.g., statistics for data, cluster status) within our algorithm to facilitate more intelligent selection of strategies to use at runtime [1]. Last but not least, our algorithmic solutions for ameliorating skew effects may find successful applications in related research problems (e.g., distributed graph query processing, graph generation, and graph indexing).

Acknowledgments. The research of YW is supported by Research Foundation Flanders (FWO) during her sabbatical visit to Hasselt University, Belgium. The research of YL, GF, JH and PD is supported by the Netherlands Organisation for Scientific Research (NWO). We thank SURFsara for their support of this investigation and Boudewijn van Dongen for his insightful comments. We thank the reviewers for their helpful suggestions.

References

1. Afrati, F.N., Sarma, A.D., Salihoglu, S., Ullman, J.D.: Upper and lower bounds on the cost of a Map-Reduce computation. CoRR, abs/1206.4377 (2012)
2. Bader, D.A., Madduri, K.: GTgraph: A suite of synthetic graph generators, `http://www.cse.psu.edu/~madduri/software/GTgraph/index.html`
3. Blom, S., Orzan, S.: A distributed algorithm for strong bisimulation reduction of state spaces. Int. J. Softw. Tools Technol. Transfer 7, 74–86 (2005)
4. Buneman, P., Grohe, M., Koch, C.: Path queries on compressed XML. In: Proc. VLDB, Berlin, Germany, pp. 141–152 (2003)
5. Clauset, A., Shalizi, C., Newman, M.: Power-law distributions in empirical data. SIAM Review 51(4), 661–703 (2009)
6. Cohen, J.: Graph twiddling in a MapReduce world. Computing in Science Engineering 11(4), 29–41 (2009)
7. de Lange, Y.: MapReduce based algorithms for localized bisimulation. Master's thesis, Eindhoven University of Technology (2013)
8. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM 51(1), 107–113 (2008)
9. Dovier, A., Piazza, C., Policriti, A.: An efficient algorithm for computing bisimulation equivalence. Theor. Comp. Sci. 311(1-3), 221–256 (2004)
10. Fan, W.: Graph pattern matching revised for social network analysis. In: Proc. ICDT, Berlin, Germany, pp. 8–21 (2012)
11. Fan, W., Li, J., Wang, X., Wu, Y.: Query preserving graph compression. In: Proc. SIGMOD, Scottsdale, AZ, USA, pp. 157–168 (2012)
12. Gufler, B., Augsten, N., Reiser, A., Kemper, A.: Handling data skew in MapReduce. In: Proc. CLOSER, pp. 574–583 (2011)

13. Gufler, B., Augsten, N., Reiser, A., Kemper, A.: Load balancing in MapReduce based on scalable cardinality estimates. In: Proc. ICDE, pp. 522–533 (2012)
14. Hadoop (2012), `http://hadoop.apache.org/`
15. Ibrahim, S., Jin, H., Lu, L., Wu, S., He, B., Qi, L.: LEEN: Locality/fairness-aware key partitioning for MapReduce in the cloud. In: CloudCom, pp. 17–24 (2010)
16. Kaushik, R., Shenoy, P., Bohannon, P., Gudes, E.: Exploiting local similarity for indexing paths in graph-structured data. In: ICDE, San Jose, pp. 129–140 (2002)
17. Kwon, Y., Ren, K., Balazinska, M., Howe, B.: Managing Skew in Hadoop. IEEE Data Eng. Bull. 36(1), 24–33 (2013)
18. Lin, J., Dyer, C.: Data-Intensive Text Processing with MapReduce. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers (2010)
19. Lin, J., Schatz, M.: Design patterns for efficient graph algorithms in MapReduce. In: Proc. MLG, Washington, D.C., pp. 78–85 (2010)
20. Luo, Y., Fletcher, G.H.L., Hidders, J., Wu, Y., De Bra, P.: I/O-efficient algorithms for localized bisimulation partition construction and maintenance on massive graphs. CoRR, abs/1210.0748 (2012)
21. Milo, T., Suciu, D.: Index structures for path expressions. In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 277–295. Springer, Heidelberg (1998)
22. Paige, R., Tarjan, R.: Three partition refinement algorithms. SIAM J. Comput. 16, 973 (1987)
23. Picalausa, F., Luo, Y., Fletcher, G.H.L., Hidders, J., Vansummeren, S.: A structural approach to indexing triples. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 406–421. Springer, Heidelberg (2012)
24. Sangiorgi, D., Rutten, J.: Advanced Topics in Bisimulation and Coinduction. Cambridge University Press, New York (2011)
25. Vernica, R., Balmin, A., Beyer, K.S., Ercegovac, V.: Adaptive MapReduce using situation-aware mappers. In: Proc. EDBT, pp. 420–431 (2012)

Sampling Estimators for Parallel Online Aggregation

Chengjie Qin and Florin Rusu

University of California, Merced
cqin3@ucmerced.edu, frusu@ucmerced.edu

Abstract. Online aggregation provides estimates to the final result of a computation during the actual processing. The user can stop the computation as soon as the estimate is accurate enough, typically early in the execution. When coupled with parallel processing, this allows for the interactive data exploration of the largest datasets. In this paper, we identify the main functionality requirements of sampling-based parallel online aggregation—partial aggregation, parallel sampling, and estimation. We argue for overlapped online aggregation as the only scalable solution to combine computation and estimation. We analyze the properties of existent estimators and design a novel sampling-based estimator that is robust to node delay and failure. When executed over a massive 8TB TPC-H instance, the proposed estimator provides accurate confidence bounds early in the execution even when the cardinality of the final result is seven orders of magnitude smaller than the dataset size and achieves linear scalability.

Keywords: parallel databases, estimation, sampling, online aggregation.

1 Introduction

Interactive data exploration is a prerequisite in model design. It requires the analyst to execute a series of exploratory queries in order to find patterns or relationships in the data. In the Big Data context, it is likely that the entire process is time-consuming even for the fastest parallel database systems given the size of the data and the sequential nature of exploration—the next query to be asked is always dependent on the previous. Online aggregation [1] aims at reducing the duration of the process by allowing the analyst to rule out the non-informative queries early in the execution. To make this possible, an estimate to the final result of the query with progressively narrower confidence bounds is continuously returned to the analyst. When the confidence bounds become tight enough, typically early in the processing, the analyst can decide to stop the execution and focus on a subsequent query.

Online aggregation in a centralized setting received a lot of attention since its introduction in the late nineties. The extension to parallel environments was mostly considered unnecessary – when considered, it was direct parallelization of serial algorithms – given the performance boost obtained in such systems by simply increasing the physical resources. With the unprecedented increase in data volumes and the proliferation of multi-core processors, parallel online aggregation becomes a necessary tool in the Big Data analytics landscape. It is the combination of parallel processing and estimation what truly makes interactive exploration of massive datasets feasible.

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 204–217, 2013.

In this paper, we identify the main requirements for parallel online aggregation—partial aggregation, parallel sampling, and estimation. Partial aggregation requires the extraction of a snapshot of the system during processing. What data are included in the snapshot is the result of parallel sampling, while estimates and confidence bounds for the query result are computed from the extracted samples. Our specific contributions are as follows:

- We discuss in details each stage in the parallel online aggregation process.
- We analyze and thoroughly compare the existent parallel sampling estimators.
- We introduce a scalable sampling estimator which exhibits increased accuracy in the face of node delay and failure.
- We provide an implementation for the proposed estimator that confirms its accuracy even for extremely selective queries over a massive 8TB TPC-H instance.

2 Preliminaries

We consider aggregate computation in a parallel cluster environment consisting of multiple processing nodes. Each processing node has a multi-core processor consisting of one or more CPUs, thus introducing an additional level of parallelism. Data are partitioned into fixed size chunks that are stored across the processing nodes. Parallel aggregation is supported by processing multiple chunks at the same time both across nodes as well as across the cores inside a node.

We focus on the computation of general SELECT-PROJECT-JOIN (SPJ) queries having the following SQL form:

$$\begin{aligned}&\texttt{SELECT SUM(f(t}_1 \bullet \texttt{t}_2\texttt{))}\\&\texttt{FROM TABLE}_1\texttt{ AS t}_1\texttt{, TABLE}_2\texttt{ AS t}_2\\&\texttt{WHERE P(t}_1 \bullet \texttt{t}_2\texttt{)}\end{aligned} \tag{1}$$

where • is the concatenation operator, f is an arbitrary *associative decomposable aggregate function* [2] over the tuple created by concatenating t_1 and t_2, and P is some boolean predicate that can embed selection and join conditions. The class of associative decomposable aggregate functions, i.e., functions that are associative and commutative, is fairly extensive and includes the majority of standard SQL aggregate functions. Associative decomposable aggregates allow for the maximum degree of parallelism in their evaluation since the computation is independent of the order in which data inside a chunk are processed as well as of the order of the chunks, while partial aggregates computed over different chunks can be combined together straightforwardly. While the paper does not explicitly discuss aggregate functions other than SUM, functions such as COUNT, AVERAGE, STD DEV, and VARIANCE can all be handled easily—they are all associative decomposable. For example, COUNT is a special case of SUM where $f(\cdot) = 1$ for any tuple, while AVERAGE can be computed as the ratio of SUM and COUNT.

Parallel aggregation. Aggregate evaluation takes two forms in parallel databases. They differ in how the partial aggregates computed for each chunk are combined together.

In the centralized approach, all the partial aggregates are sent to a common node – the coordinator – that is further aggregating them to produce the final result. As an intermediate step, local aggregates can be first combined together and only then sent to the coordinator. In the parallel approach, the nodes are first organized into an aggregation tree. Each node is responsible for aggregating its local data and the data of its children. The process is executed level by level starting from the leaves, with the final result computed at the root of the tree. The benefit of the parallel approach is that it also parallelizes the aggregation of the partial results across all the nodes rather than burdening a single node (with data and computation). The drawback is that in the case of a node failure it is likely that more data are lost. Notice that these techniques are equally applicable inside a processing node, at the level of a multi-core processor.

Online aggregation. The idea in online aggregation is to compute only an estimate of the aggregate result based on a sample of the data [1]. In order to provide any useful information though, the estimate is required to be accurate and statistically significant. Different from one-time estimation [3] that might produce very inaccurate estimates for arbitrary queries, online aggregation is an iterative process in which a series of estimators with improving accuracy are generated. This is accomplished by including more data in estimation, i.e., increasing the sample size, from one iteration to another. The end-user can decide to run a subsequent iteration based on the accuracy of the estimator. Although the time to execute the entire process is expected to be much shorter than computing the aggregate over the entire dataset, this is not guaranteed, especially when the number of iterations is large. Other issues with *iterative online aggregation* [4,5] regard the choice of the sample size and reusing the work done across iterations.

An alternative that avoids these problems altogether is to completely *overlap query processing with estimation* [6,7]. As more data are processed towards computing the final aggregate, the accuracy of the estimator improves accordingly. For this to be true though, data are required to be processed in a statistically meaningful order, i.e., random order, to allow for the definition and analysis of the estimator. This is typically realized by randomizing data during the loading process. The drawback of the overlapped approach is that the same query is essentially executed twice—once towards the final aggregate and once for computing the estimator. As a result, the total execution time in the overlapped case is expected to be higher when compared to the time it takes to execute each task separately.

3 Parallel Online Aggregation

There are multiple aspects that have to be considered in the design of a parallel online aggregation system. First, a mechanism that allows for the computation of partial aggregates has to be devised. Second, a parallel sampling strategy to extract samples from data over which partial aggregates are computed has to be designed. Each sampling strategy leads to the definition of an estimator for the query result that has to be analyzed in order to derive confidence bounds. In this section, we discuss in detail each of these aspects for the overlapped online aggregation approach.

3.1 Partial Aggregation

The first requirement in any online aggregation system is a mechanism to compute partial aggregates over some portion of the data. Partial aggregates are typically a superset of the query result since they have to contain additional data required for estimation. The partial aggregation mechanism can take two forms. We can fix the subset of the data used in partial aggregation and execute a normal query. Or we can interfere with aggregate computation over the entire dataset to extract partial results before the computation is completed. The first alternative corresponds to iterative online aggregation, while the second to overlapped execution.

Partial aggregation in a parallel setting raises some interesting questions. For iterative online aggregation, the size and location of the data subset used to compute the partial aggregate have to be determined. It is common practice to take the same amount of data from each node in order to achieve load balancing. Or to have each node process a subset proportional to its data as a fraction from the entire dataset. Notice though that it is not necessary to take data from all the nodes. In the extreme case, the subset considered for partial aggregation can be taken from a single node. Once the data subset at each node is determined, parallel aggregation proceeds normally, using either the centralized or parallel strategy. In the case of overlapped execution, a second process that simply aggregates the current results at each node has to be triggered whenever a partial aggregate is computed. The aggregation strategy can be the same or different from the strategy used for computing the final result. Centralized aggregation might be more suitable though due to the reduced interference. The amount of data each node contributes to the result is determined only by the processing speed of the node. Since the work done for partial aggregation is also part of computing the final aggregate, it is important to reuse the result so that the overall execution time is not increased unnecessarily.

3.2 Parallel Sampling

In order to provide any information on the final result, partial aggregates have to be statistically significant. It has to be possible to define and analyze estimators for the final result using partial aggregates. Online aggregation imposes an additional requirement. The accuracy of the estimator has to improve when more data are used in the computation of partial aggregates. In the extreme case of using the entire dataset to compute the partial aggregate, the estimator collapses on the final result. The net effect of these requirements is that the data subset on which the partial aggregate is computed cannot be arbitrarily chosen. Since sampling satisfies these requirements, the standard approach in online aggregation is to choose the subset used for partial aggregation as a random sample from the data. Thus, an important decision that has to be taken when designing an online aggregation system is how to generate random samples.

Centralized sampling. According to the literature [8], there are two methods to generate samples from the data in a centralized setting. The first method is based on using an index that provides the random order in which to access the data. While it does not require any pre-processing, this method is highly inefficient due to the large number of random accesses to the disk. The second method is based on the idea of storing data in random order on disk such that a sequential scan returns random samples at any position. Although this method requires considerable pre-processing at loading time to permute data randomly, it is the preferred randomization method in online aggregation systems since the cost is paid only once and it can be amortized over the execution of multiple queries—the indexing method incurs additional cost for each query.

Sampling synopses. It is important to make the distinction between the runtime sampling methods used in online aggregation and estimation based on static samples taken offline [3], i.e., sampling synopses. In the later case, a sample of fixed size is taken only once and all subsequent queries are answered using the sample. This is typically faster than executing sampling at runtime, during query processing. The problem is that there are queries that cannot be answered from the sample accurately enough, for example, highly selective queries. The only solution in this case is to extract a larger sample entirely from scratch which is prohibitively expensive. The sampling methods for online aggregation avoid this problem altogether due to their incremental design that degenerates in a sample consisting of the entire dataset in the worst case.

Sample size. Determining the correct sample size to allow for accurate estimations is an utterly important problem in the case of sampling synopses and iterative online aggregation. If the sample size is not large enough, the entire sampling process has to be repeated, with unacceptable performance consequences. While there are methods that guide the selection of the sample size for a given accuracy in the case of a single query, they require estimating the variance of the query estimator—an even more complicated problem. In the case of overlapped online aggregation, choosing the sample size is not a problem at all since the entire dataset is processed in order to compute the correct result. The only condition that has to be satisfied is that the data seen up to any point during processing represent a sample from the entire dataset. As more data are processed towards computing the query result, the sample size increases automatically. Both runtime sampling methods discussed previously satisfy this property.

Stratified sampling. There are multiple alternatives to obtain a sample from a partitioned dataset—the case in a parallel setting. The straightforward solution is to consider each partition independently and to apply centralized sampling algorithms inside the partition. This type of sampling is known as *stratified sampling* [9]. While stratified sampling generates a random sample for each partition, it is not guaranteed that when putting all the local samples together the resulting subset is a random sample from the entire data. For this to be the case, it is required that the probability of a tuple to be in the sample is the same across all the partitions. The immediate solution to this problem is to take local samples that are proportional with the partition size.

Global randomization. A somehow more complicated solution is to make sure that a tuple can reside at any position in any partition—*global randomization*. This can be achieved by randomly shuffling the data across all the nodes—as a direct extension of the similar centralized approach. The global randomization process consists of two stages, each executed in parallel at every node. In the first stage, each node partitions the local data into sets corresponding to all the other nodes in the environment. In the second stage, each node generates a random permutation of the data received from all the other nodes—random shuffling. This is required in order to separate the items received from the same origin.

The main benefit provided by global randomization is that it simplifies the complexity of the sampling process in a highly-parallel asynchronous environment. This in turn allows for compact estimators to be defined and analyzed—a single estimator across the entire dataset. It also supports more efficient sampling algorithms that require a reduced level of synchronization, as is the case with our estimator. Moreover, global randomization has another important characteristic for online aggregation—it allows for incremental sampling. What this essentially means is that in order to generate a sample of a larger size starting from a given sample is enough to obtain a sample of the remaining size. It is not even required that the two samples are taken from the same partition since random shuffling guarantees that a sample taken from a partition is actually a sample from the entire dataset. Equivalently, to get a sample from a partitioned dataset after random shuffling, it is not necessary to get a sample from each partition.

While random shuffling in a centralized environment is a time-consuming process executed in addition to data loading, global randomization in a parallel setting is a standard hash-based partitioning process executed as part of data loading. Due to the benefits provided for workload balancing and for join processing, hash-based partitioning is heavily used in parallel data processing even without online aggregation. Thus, we argue that global randomization for parallel online aggregation is part of the data loading process and it comes at virtually no cost with respect to sampling.

3.3 Estimation

While designing sampling estimators for online aggregation in a centralized environment is a well-studied problem, it is not so clear how these estimators can be extended to a highly-parallel asynchronous system with data partitioned across nodes. To our knowledge, there are two solutions to this problem proposed in the literature. In the first solution, a sample over the entire dataset is built from local samples taken independently at each partition. An estimator over the constructed sample is then defined. We name this approach *single estimator*. In the single estimator approach, the fundamental question is how to generate a single random sample of the entire dataset from samples extracted at the partition level. The strategy proposed in [4] requires synchronization between all the sampling processes executed at partition level in order to guarantee that the same fraction of the data are sampled at each partition. To implement this strategy, serialized access to a common resource is required for each item processed. This results in unacceptable execution time increase when estimation is active.

In the second solution, which we name *multiple estimators*, an estimator is defined for each partition. As in stratified sampling theory [9], these estimators are then combined into a single estimator over the entire dataset. The solution proposed in [10] follows this approach. The main problem with the multiple estimators strategy is that the final result computation and the estimation are separate processes with different states that require more complicated implementation.

We propose an asynchronous sampling estimator specifically targeted at parallel online aggregation that combines the advantages of the existing strategies. We define our estimator as in the single estimator solution, but without the requirement for synchronization across the partition-level sampling processes which can be executed independently. This results in much better execution time. When compared to the multiple estimators approach, our estimator has a much simpler implementation since there is complete overlap between execution and estimation. In this section, we analyze the properties of the estimator and compare it with the two estimators it inherits from. Then, in Section 4 we provide insights into the actual implementation, while in Section 5 we present experimental results to evaluate the accuracy of the estimator and the runtime performance of the estimation.

Generic Sampling Estimator. To design estimators for the parallel aggregation problem we first introduce a generic sampling estimator for the centralized case. This is a standard estimator based on sampling without replacement [9] that is adequate for online aggregation since it provides progressively increasing accuracy. We define the estimator for the simplified case of aggregating over a single table and then show how it can be generalized to `GROUP BY` and general `SPJ` queries.

Consider the dataset D to have a single partition sorted in random order. The number of items in D (size of D) is $|D|$. While sequentially scanning D, any subset $S \subseteq D$ represents a random sample of size $|S|$ taken without replacement from D. We define an estimator for the SQL aggregate in Eq. 1 as follows:

$$X = \frac{|D|}{|S|} \sum_{s \in S, \mathtt{P}(s)} \mathtt{f}(s) \tag{2}$$

where `f` and `P` are the aggregate function and the boolean predicate embedding selection and join conditions, respectively. X has the properties given in Lemma 1:

Lemma 1. *X is an unbiased estimator for the aggregation problem, i.e., $E[X] = \sum_{d \in D, \mathtt{P}(d)} \mathtt{f}(d)$, where $E[X]$ is the expectation of X. The variance of X is equal to:*

$$Var(X) = \frac{|D| - |S|}{(|D| - 1)|S|} \left[|D| \sum_{d \in D, \mathtt{P}(d)} \mathtt{f}^2(d) - \left(\sum_{d \in D, \mathtt{P}(d)} \mathtt{f}(d) \right)^2 \right] \tag{3}$$

It is important to notice the factor $|D| - |S|$ in the variance numerator which makes the variance to decrease while the size of the sample increases. When the sample is the entire dataset, the variance becomes zero, thus the estimator is equal to the exact query result. The standard approach to derive confidence bounds [11,12,13] is to assume a normal distribution for estimator X with the first two frequency moments given by $E[X]$ and $\mathrm{Var}(X)$. The actual bounds are subsequently computed at the required confidence level from the cumulative distribution function (cdf) of the normal distribution. Since the width of the confidence bounds is proportional with the variance, a decrease in the variance makes the confidence bounds to shrink. If the normality condition does not hold, more conservative distribution-independent confidence bounds can be derived using the Chebyshev-Chernoff inequalities, for example.

A closer look at the variance formula in Eq. 3 reveals the dependency on the entire dataset D through the two sums over all the items $d \in D$ that satisfy the selection predicate `P`. Unfortunately, when executing the query we have access only to the sampled data. Thus, we need to compute the variance from the sample. We do this by defining a variance estimator, $\text{Est}_{\text{Var}(X)}$, with the following properties:

Lemma 2. *The estimator*

$$Est_{Var(X)} = \frac{|D|(|D| - |S|)}{|S|^2(|S| - 1)} \left[|S| \sum_{s \in S, P(s)} f^2(s) - \left(\sum_{s \in S, P(s)} f(s) \right)^2 \right] \tag{4}$$

is an unbiased estimator for the variance in Eq. 3.

Having the two estimators X and $\text{Est}_{\text{Var}(X)}$ computed over the sample S, we are in the position to provide the confidence bounds required by online aggregation in a centralized environment. The next step is to extend the generic estimators to a parallel setting where data are partitioned across multiple processing nodes.

Before that though, we discuss on how to extend the generic estimator to `GROUP BY` and general `SPJ` queries. For `GROUP BY`, a pair of estimators X and $\text{Est}_{\text{Var}(X)}$ can be defined independently for each group. The only modification is that predicate `P` includes an additional selection condition corresponding to the group. A detailed analysis on how X and $\text{Est}_{\text{Var}(X)}$ can be extended to general `SPJ` queries is given in [11]. The main idea is to include the join condition in predicate `P` and take into consideration the effect it has on the two samples. We do not provide more details since the focus of this paper is on parallel versions of X and $\text{Est}_{\text{Var}(X)}$.

Single Estimator Sampling. When the dataset D is partitioned across N processing nodes, i.e., $D = D_1 \cup D_2 \cup \cdots \cup D_N$, a sample $S_i, 1 \leq i \leq N$ is taken independently at each node. These samples are then put together in a sample $S = S_1 \cup S_2 \cup \cdots \cup S_N$ over the entire dataset D. To guarantee that S is indeed a sample from D, in the case of the synchronized estimator in [4] it is enforced that the sample ratio $\frac{S_i}{D_i}$ is the same across all the nodes. For the estimator we propose, we let the nodes run independently and only during the partial aggregation stage we combine the samples from all the nodes as S. Thus, nodes operate asynchronously at different speed and produce samples with different size. Global randomization guarantees though that the combined sample S is indeed a sample over the entire dataset. As a result, the generic sampling estimator in Eq. 2 can be directly applied without any modifications.

Multiple Estimators Sampling. For the multiple estimators strategy, the aggregate $\sum_{d \in D, \mathtt{P}(d)} \mathtt{f}(d)$ can be decomposed as $\sum_{i=1}^{N} \sum_{d \in D_i, \mathtt{P}(d)} \mathtt{f}(d)$, with each node computing the sum over the local partition in the first stage followed by summing-up the local results to get the overall result in the second stage. An estimator is defined for each partition as $X_i = \frac{|D_i|}{|S_i|} \sum_{s \in S_i, \mathtt{P}(s)} \mathtt{f}(s)$ based on the generic sampling estimator in Eq. 2. We can then immediately infer that the sum of the estimators X_i, $\sum_{i=1}^{N} X_i$, is an unbiased estimator for the query result and derive the variance $\text{Var}\left(\sum_{i=1}^{N} X_i\right) = \sum_{i=1}^{N} \text{Var}\left(X_i\right)$ if the sampling process across partitions is independent. Since the samples are taken independently from each data partition, local data randomization at each processing node is sufficient for the analysis to hold.

Discussion. We propose an estimator for parallel online aggregation based on the *single estimator* approach. The main difference is that our estimator is completely asynchronous and allows fully parallel evaluation. We show how it can be derived and analyzed starting from a generic sampling estimator for centralized settings. We conclude with a detailed comparison with a stratified sampling estimator (or *multiple estimators*) along multiple dimensions:

Data randomization. While the multiple estimators approach requires only local randomization, the single estimator approach requires global randomization across all the nodes in the system. Although this might seem a demanding requirement, the randomization process can be entirely overlapped with data loading as part of hash-based data partitioning.

Dataset information. Multiple estimators requires each node to have knowledge of the local partition cardinality, i.e., $|D_i|$. Single estimator needs only full cardinality information, i.e., $|D|$, where the estimation is invoked.

Accuracy. According to the stratified sampling theory, multiple estimators provides better accuracy when the size of the sample at each node is proportional with the local dataset size [9]. This is not true in the general case though with the variance of the estimators being entirely determined by the samples at hand. In a highly asynchronous parallel setting, this optimal condition is hard to enforce.

Convergence rate. As with accuracy, it is not possible to characterize the relative convergence rate of the two methods in the general case. Nonetheless, we can argue that multiple estimators is more sensitive to discrepancies in processing across the nodes since the effect on variance is only local. Consider for example the case when one variance is considerably smaller than the others. Its effect on the overall variance is asymptotically limited by the fraction it represents from the overall variance rather than the overall variance.

Fault tolerance. The effect of node failure is catastrophic for multiple estimators. If one node cannot be accessed, it is impossible to compute the estimator and provide bounds since the corresponding variance is infinite. For single estimator, the variance decrease stops at a higher value than zero. This results in bounds that do not collapse on the true result even when the processing concludes.

4 Implementation

We implement the sampling estimators for online aggregation in GLADE [2,14], a parallel processing system optimized for the execution of associative-decomposable User-Defined Aggregates (UDA). In this section, we discuss the most significant extensions made to the GLADE framework in order to support online aggregation. Then, we present the implementation of the single estimator as an example UDA.

Extended UDA Interface. Table 1 summarizes the extended UDA interface we propose for parallel online aggregation. This interface abstracts aggregation and estimation

Table 1. Extended UDA interface

Method	Usage
Init () Accumulate (Item d) Merge (UDA $input_1$,UDA $input_2$, UDA $output$) Terminate ()	Basic interface
Serialize () Deserialize ()	Transfer UDA across processes
EstTerminate () EstMerge (UDA $input_1$,UDA $input_2$, UDA $output$)	Partial aggregate computation
Estimate ($estimator$, $lower$, $upper$, $confidence$)	Online estimation

in a reduced number of methods, releasing the user from the details of the actual execution in a parallel environment which are taken care of transparently by GLADE. Thus, the user can focus only on estimation modeling.

The first extension is specifically targeted at estimation modeling for online aggregation. To support estimation, the UDA state needs to be enriched with additional data on top of the original aggregate. Although it is desirable to have a perfect overlap between the final result computation and estimation, this is typically not possible. In the few situations when it is possible, no additional changes to the UDA interface are required. For the majority of the cases though, the UDA interface needs to be extended in order to distinguish between the final result and a partial result used for estimation. There are at least two methods that need to be added to the UDA interface—`EstTerminate` and `EstMerge`. `EstTerminate` computes a local estimator at each node. It is invoked after merging the local UDAs during the estimation process. `EstMerge` is called to put together in a single UDA the estimators computed at each node by `EstTerminate`. It is invoked with UDAs originating at different nodes. Notice that `EstTerminate` is an intra-node method while `EstMerge` is inter-node. It is possible to further separate the estimation from aggregate computation and have an intra-node `EstMerge` and an inter-node `EstTerminate`.

The second extension to the UDA interface is the `Estimate` method. It is invoked by the user application on the UDA returned by the framework as a result of an estimation request. The complexity of this method can range from printing the UDA state to complex statistical models. In the case of online aggregation, `Estimate` computes an estimator for the aggregate result and corresponding confidence bounds.

Example UDA. We present the UDA corresponding to the proposed asynchronous estimator for single-table aggregation – more diverse examples of higher complexity are presented in [15] – having the following SQL form:

`SELECT SUM(f(t)) FROM TABLE AS t WHERE P(t)` (5)

which computes the SUM of function `f` applied to each tuple in table `TABLE` that satisfies condition `P`. It is straightforward to express this aggregate in UDA form. The `state` consists only of the running sum, initialized at zero. `Accumulate` updates the

Algorithm 1. *UDASum-SingleEstimator*

```
State: $sum$; $sumSq$; $count$
Init ()
  $sum = 0$; $sumSq = 0$; $count = 0$
Accumulate (Tuple t)
  if P(t) then
      $sum = sum + \mathtt{f}(\mathtt{t})$; $sumSq = sumSq + \mathtt{f}^2(\mathtt{t})$; $count = count + 1$
  end if
Merge (UDASum $input_1$, UDASum $input_2$, UDASum $output$)
  $output.sum = input_1.sum + input_2.sum$
  $output.sumSq = input_1.sumSq + input_2.sumSq$
  $output.count = input_1.count + input_2.count$
Terminate ()
Estimate ($estimator$, $lowerBound$, $upperBound$, $confLevel$)
  $estimator = \frac{|D|}{count} * sum$
  $estVar = \frac{|D|*(|D|-count)}{count^2*(count-1)} * \left(count * sumSq - sum^2\right)$
  $lowerBound = estimator + NormalCDF\left(\frac{1-confLevel}{2}, \sqrt{estVar}\right)$
  $upperBound = estimator + NormalCDF\left(confLevel + \frac{1-confLevel}{2}, \sqrt{estVar}\right)$
```

current sum with `f(t)` only for the tuples `t` satisfying the condition `P`, while `Merge` adds the states of the input UDAs and stores the result as the state of the output UDA.

UDASum-SingleEstimator implements the estimator we propose. No modifications to the UDA interface are required. Looking at the UDA state, it might appear erroneous that no sample is part of the state when a sample over the entire dataset is required in the estimator definition. Fortunately, the estimator expectation and variance can be derived from the three variables in the state computed locally at each node and then merged together globally. This reduces dramatically the amount of data that needs to be transferred between nodes. To compute the estimate and the bounds, knowledge of the full dataset size is required in `Estimate`.

Parallel Online Aggregation in GLADE. At a high level, enhancing GLADE with online aggregation is just a matter of providing support for UDAs expressed using the extended UDA interface in Table 1 in order to extract a snapshot of the system state that can be used for estimation. While this is a good starting point, there are multiple aspects that require careful consideration. For instance, the system is expected to process partial result requests at any rate, at any point during query execution, and with the least amount of synchronization among the processing nodes. Moreover, the system should not incur any overhead on top of the normal execution when online aggregation is enabled. Under these requirements, the task becomes quite challenging.

Our solution overlaps online estimation and actual query processing at all levels of the system and applies multiple optimizations. Abstractly, this corresponds to executing two simultaneous UDA computations. Rather than treating actual computation and estimation as two separate UDAs, we group everything into a single UDA satisfying the extended interface. More details can be found in an extended version of the paper [15].

5 Empirical Evaluation

We present experiments that compare the asynchronous single estimator we propose in this paper and the multiple estimators approach. We evaluate the "time 'til utility" (TTU) [13] or convergence rate of the estimators and the scalability of the estimation process on a 9-node shared nothing cluster—one node is configured as the coordinator and the other 8 nodes are workers. The dataset used in our experiments is TPC-H scale **8,000** (**8*TB***)—each node stores 1*TB*. For more details on the experimental setup, we refer the reader to our extended report [15].

The aggregation task we consider is given by the following general SPJ query:

```
SELECT n_name, SUM(l_extendprice*(1-l_discount)*(1+l_tax))
FROM lineitem, supplier, nation
WHERE l_shipdate = 1993-02-26 AND l_quantity = 1 AND
 l_discount between [0.02,0.03] AND
 l_suppkey = s_suppkey AND s_nationkey = n_nationkey
GROUP BY n_name
```

To execute the query in parallel, supplier and nation are replicated across all the nodes. They are loaded in memory, pre-joined, and hashed on s_suppkey. lineitem is scanned sequentially and the matching tuple is found and inserted in the group-by hash table. Merging the GLA states proceeds as in the group-by case. This join strategy is common in parallel databases.

What is important to notice about this query is the extremely high selectivity of the selection predicates. Out of the 48×10^9 tuples in lineitem, only $35,000$ tuples are part of the result. These tuples are further partitioned by the GROUP BY clause such that the number of tuples in each group is around $1,500$. This corresponds to a selectivity of 29×10^{-7}—a veritable needle in the haystack query. Providing sampling estimates for so highly selective queries is a very challenging task.

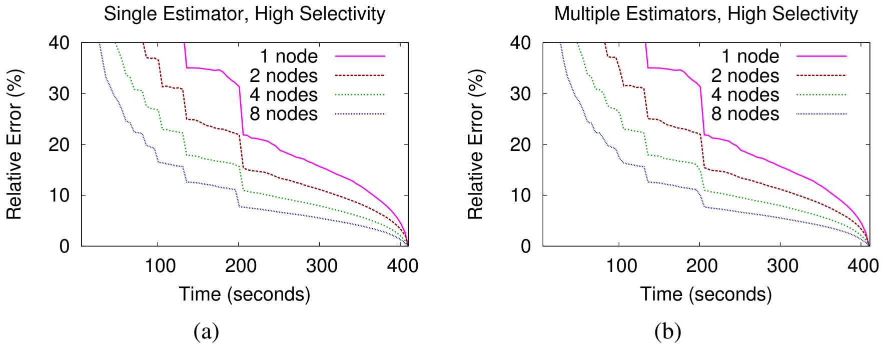


Fig. 1. Comparison between single estimator and multiple estimators. The plots print the results corresponding to the PERU group.

The results are depicted in Figure 1. As expected, the accuracy of the two estimators increases as more data are processed, converging on the correct result in the end. The effect of using a larger number of processing nodes is also clear. With 8 nodes more result tuples are discovered in the same amount of time, thus the better accuracy. Since the query takes the same time when proportionally more data and processing nodes are used, the scaleup of the entire process is also confirmed. What is truly remarkable though, is the reduced TTU even for this highly selective query. Essentially, the error is already under 10% when less than half of the data are processed. The reason for this is the effective tuple discovery process amplified by parallel processing.

When comparing the two estimators, there is no much difference—both in accuracy and in execution time. This confirms the effectiveness of the proposed estimator since the multiple estimators approach is known to have optimal accuracy in this particularly balanced scenario. It is also important to notice that the execution time is always limited by the available I/O throughput. The difference between the two estimators is clear when straggler nodes are present or when nodes die. Essentially, no estimate can be computed by the multiple estimators approach when any node dies. We refer the reader to the extended version of the paper [15] for experiments concerning the reliability of the estimators—and many other empirical evaluations.

6 Related Work

There is a plethora of work on online aggregation published in the database literature starting with the seminal paper by Hellerstein et al. [1]. We can broadly categorize this body of work into system design [16,6], online join algorithms [17,11,18], and methods to derive confidence bounds [17,11,12]. All of this work is targeted at single-node centralized environments.

The parallel online aggregation literature is not as rich though. We identified only three lines of research that are closely related to this paper. Luo et al. [10] extend the centralized ripple join algorithm [17] to a parallel setting. A stratified sampling estimator [9] is defined to compute the result estimate while confidence bounds cannot always be derived. This is similar to the multiple estimators approach. Wu et al. [4] extend online aggregation to distributed P2P networks. They introduce a synchronized sampling estimator over partitioned data that requires data movement from storage nodes to processing nodes. This corresponds to the synchronized single estimator solution.

The third piece of relevant work is online aggregation in Map-Reduce. In [19], stock Hadoop is extended with a mechanism to compute partial aggregates. In subsequent work [7], an estimation framework based on Bayesian statistics is proposed. BlinkDB [20] implements a multi-stage approximation mechanism based on precomputed sampling synopses of multiple sizes, while EARL [5] is an iterative online aggregation system that uses bootstrapping to produce multiple estimators from the same sample. Our focus is on sampling estimators for overlapped online aggregation. This is a more general problem that subsumes sampling synopses and estimators for iterative online aggregation.

7 Conclusions

We propose the combination of parallel processing and online aggregation as a feasible solution for Big Data analytics. We identify the main stages – partial aggregation, parallel sampling, and estimation – in the online aggregation process and discuss how they can be extended to a parallel environment. We design a scalable sampling-based estimator with increased accuracy in the face of node delay and failure. We implement the estimator in GLADE [2] – a highly-efficient parallel processing system – to confirm its accuracy even for extremely selective queries over a massive TPC-H 8TB instance.

Acknowledgments. This work was supported in part by a gift from LogicBlox.

References

1. Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online Aggregation. In: SIGMOD (1997)
2. Rusu, F., Dobra, A.: GLADE: A Scalable Framework for Efficient Analytics. Operating Systems Review 46(1) (2012)
3. Cormode, G., Garofalakis, M.N., Haas, P.J., Jermaine, C.: Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. Foundations and Trends in Databases 4(1-3) (2012)
4. Wu, S., Jiang, S., Ooi, B.C., Tan, K.L.: Distributed Online Aggregation. PVLDB 2(1) (2009)
5. Laptev, N., Zeng, K., Zaniolo, C.: Early Accurate Results for Advanced Analytics on MapReduce. PVLDB 5(10) (2012)
6. Rusu, F., Xu, F., Perez, L.L., Wu, M., Jampani, R., Jermaine, C., Dobra, A.: The DBO Database System. In: SIGMOD (2008)
7. Pansare, N., Borkar, V.R., Jermaine, C., Condie, T.: Online Aggregation for Large MapReduce Jobs. PVLDB 4(11) (2011)
8. Olken, F.: Random Sampling from Databases. Ph.D. thesis, UC Berkeley (1993)
9. Cochran, W.G.: Sampling Techniques. Wiley (1977)
10. Luo, G., Ellmann, C.J., Haas, P.J., Naughton, J.F.: A Scalable Hash Ripple Join Algorithm. In: SIGMOD (2002)
11. Jermaine, C., Dobra, A., Arumugam, S., Joshi, S., Pol, A.: The Sort-Merge-Shrink Join. TODS 31(4) (2006)
12. Jermaine, C., Arumugam, S., Pol, A., Dobra, A.: Scalable Approximate Query Processing with the DBO Engine. In: SIGMOD (2007)
13. Dobra, A., Jermaine, C., Rusu, F., Xu, F.: Turbo-Charging Estimate Convergence in DBO. PVLDB 2(1) (2009)
14. Cheng, Y., Qin, C., Rusu, F.: GLADE: Big Data Analytics Made Easy. In: SIGMOD (2012)
15. Qin, C., Rusu, F.: PF-OLA: A High-Performance Framework for Parallel On-Line Aggregation. CoRR abs/1206.0051 (2012)
16. Avnur, R., Hellerstein, J.M., Lo, B., Olston, C., Raman, B., Raman, V., Roth, T., Wylie, K.: CONTROL: Continuous Output and Navigation Technology with Refinement On-Line. In: SIGMOD (1998)
17. Haas, P.J., Hellerstein, J.M.: Ripple Joins for Online Aggregation. In: SIGMOD (1999)
18. Chen, S., Gibbons, P.B., Nath, S.: PR-Join: A Non-Blocking Join Achieving Higher Early Result Rate with Statistical Guarantees. In: SIGMOD (2010)
19. Condie, T., Conway, N., Alvaro, P., Hellerstein, J.M., Elmeleegy, K., Sears, R.: MapReduce Online. In: NSDI (2010)
20. Agarwal, S., Panda, A., Mozafari, B., Iyer, A.P., Madden, S., Stoica, I.: Blink and It's Done: Interactive Queries on Very Large Data. PVLDB 5(12) (2012)

The Business Network Data Management Platform

Daniel Ritter

Technology Development – Process and Network Integration, SAP AG,
Dietmar-Hopp-Allee 16, 69190 Walldorf
daniel.ritter@sap.com

Abstract. The discovery, representation and reconstruction of Business Networks (BN) from Network Mining (NM) raw data is a difficult problem for enterprises. This is due to huge amounts of fragmented data representing complex business processes within and across enterprise boundaries and heterogeneous technology stacks. To remain competitive, the visibility into the enterprise and partner networks on different, interrelated abstraction levels is desirable. We show the data management capabilities of a novel data discovery, mining and network inference system, called Business Network System (BNS) that reconstructs the BN - integration and business process networks - from raw data, hidden in the enterprises' landscapes. The paper covers both the foundation and key data management characteristics of BNS.

Keywords: Data Management, Data Provenance, Data System Architecture, Network Mining, Network Reconstruction.

1 Introduction

Enterprises are part of value chains consisting of business processes connecting intra- and inter-enterprise participants. The network that connects these participants with their technical, social and business relations is called a *Business Network* (BN). Even though this network is very important for the enterprise, there are few - if any - people in the organization who understand this network as the relevant data is hidden in heterogeneous enterprise system landscapes. Yet simple questions about the network (e.g., which business processes require which interfaces, which integration artifacts are obsolete) remain difficult to answer, which makes the operation and lifecycle management like data migration, landscape optimization and evolution hard and more expensive increasing with the number of the systems. To change that, *Network Mining* (NM) systems are used to discover and extract raw data [11] - be it technical data (e.g. configurations of integration products like *Enterprise Service Bus* (ESB) [6]) or business data (e.g., information about a supplier in a *Supplier Relationship Management* (SRM) product). The task at hand is to provide a system, that automatically discovers and reconstructs the "as-is" BN from the incomplete, fragmented, cross-domain NM data and make it accessible for visualization and analysis.

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 218–227, 2013.

Previous work on NM systems [11] and their extension towards a holistic management of BN [13] provide a comprehensive, theoretical foundation on how to build a system suited to this task. With the *Business Network System* (BNS), we are exploiting this opportunity stated by these requirements. In particular, we are leveraging the work on the modeling and reconstruction of integration networks [15], conformance checking of implicit data models [14], the client API [12] and the BN model [10] to deliver an emergent, holistic foundation for a declarative BN management system.

In this work we discuss the data management requirements of the BNS and shed light into its internal mechanics. The major contributions of this work are (1) a sound list of the most important requirements of the data management in the BNS, building on previous work, (2) a data processing approach suitable for these requirements, and (3) a system implementing this architecture for continuous and scalable end-to-end network query, traversal and update processing based on the data transformation and provenance approach.

Section 2 guides from the theoretical work conducted in the areas of NM [11] and *Business Network Management* (BNM) [13] to the real-world data management requirements of a BNS (refers to (1)) and sketches a high-level view on the system's architecture (refers to (3)). Section 3 provides an overview of BNS's query and update processing (refers to (2)). Section 4 reviews and discusses related work and systems that influenced BNS. Section 5 concludes the paper and lists some of the future work.

2 The Business Network System

The BN consists of a set of interrelated perspectives of domain networks (e.g., business process, integration, social), that provide a contextualized view on which business processes (i.e., business perspective) are currently running, implemented on which integration capabilities (i.e., integration perspective) and operated by whom (i.e., social perspective). To compute the BN, Network Mining (NM) systems automatically discover raw data from the enterprise landscapes [11]. These conceptual foundations are extended to theoretically ground the new BN data management domain [13].

2.1 The Data Management Requirements and Architecture

The fundamental requirements and capabilities of a BNS are derived from the theoretical foundations in previous work. In a nutshell they cover *REQ-1* the (semi-)automatic discovery of data within the enterprise landscapes and cloud applications, *REQ-2* a common, domain independent, network inference model, *REQ-3* the transformation of the domain data into this model, *REQ-4* a scalable, continuously running and declaratively programmable inference system, *REQ-5* the cross-domain and enterprise/ tenant reconstruction, *REQ-6* the ability to check the data quality and compliance to the inference model, and *REQ-7* the visualization of different perspectives (i.e., views) on the BN (e.g., business process, integration).

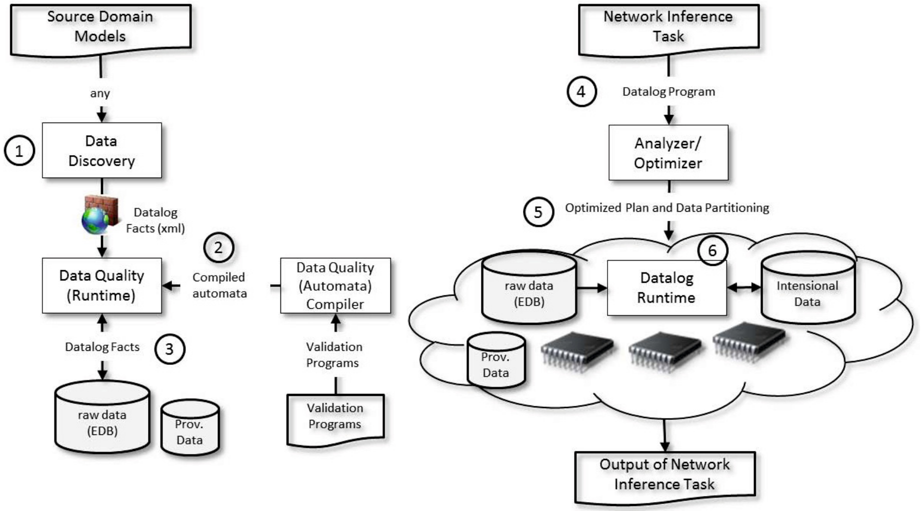


Fig. 1. System stack for BN discovery, conformance checking and continuous network reconstruction

When starting with a system, which fulfills these requirements, the specific data management and data processing aspects of the BNS summarize to the following: *REQ-8* The common access pattern for query, traversal and full-text search across the interconnected BN perspectives shall be supported by a scalable (remote) client API (e.g., through index creation) (from [12]); *REQ-9* The (remote) API shall allow efficient resource access to the BN mainly for low latency, read-only requests, "get all" and selective queries. While the availability is crucial, the consistency model for concurrent BN access could be eventual consistency (from [12]); *REQ-10* The user shall be able to enrich (e.g., labeling, grouping) and to enhance the BN data (e.g., adding/ removing nodes and edges) through the client API, while delta-changes from existing data sources and initial loads from new data sources are merged into the existing BN (from [13]); *REQ-11* Through the whole system, the data origin shall be tracked through all transformations from the source to the BN (i.e., data provenance). This shall allow for continuous data source integration, user enrichments/ enhancements as well as possible re-deployment from the BN to the data sources (from [13]); *REQ-12* The source data shall be available at all times for continuous re-computation of the network (i.e., even if the original source is not accessible for a while) (from [13] and *REQ-11*); *REQ-13* The system shall be able to process data from a growing number of data sources across different enterprises (from [13]). The data required for the reconstruction of the BN is assumed to be found in the enterprise system landscapes. However, in reality the data is fragmented and scattered across different applications and domains (e.g., log files, system landscape directories, configuration). For a single domain, the discovery (see *REQ-1*) extracts several hundreds of thousands of facts (i.e., without real-time instance data). Hence, in an enterprise context the amount of data could easily reach several ten or hundred million facts.

To sketch an idea on what these requirements mean for the construction of a BNS, Figures 1 and 2 provide a high-level view on the core data management capabilities of our BNS. Within the enterprise landscapes (not shown), source domain models consist of a mix of business process, social and integration artifacts stored in databases, packaged applications, system landscape directories (e.g., SAP SLD [17]), middleware systems (e.g., SAP PI [16]), documents/ files, application back-ends, and so on. ① When pointed to an enterprise data source through configuration by a domain expert, the BN discovery introspects the source's metadata (e.g., WSDL file for web service), discovers and transforms the domain data to a common representation (see *REQ-1,2*). The common representation, referred to as the inference model, is a uniform, formalization of the enterprise's data sources as Datalog facts (see *REQ-2,3*). The center of Figure 1 shows the core elements of a NM system, theoretically discussed in [11] and [13], which computes the perspectives of the BN for client access (see Figure 2 ⑨; *REQ-7*). After the loaded data has been checked for conformance to the inference model in ② (see *REQ-6*) it is stored as raw data ③ for the continuously running network reconstruction programs using logic programming (i.e., our approach uses Datalog due to the rationale in [15]) in ⑥ (see *REQ-4*). For that, the Datalog programs are loaded in ④ and optimally distributed across processing resources in ⑤. Since BN reconstruction works on cross-domain and the enterprise data, and (cloud) applications want to access the BN data, the NM-part of the system is located in the private or public cloud, while the discovery-part is located in the enterprise system landscapes (see *REQ-5*). That means, the data sources are highly distributed and the permanent, efficient access is not guaranteed. For that, the source data is copied to the Business Network Server by a set of protocol adapters. The data is stored as raw data, but linked to its original source for later reference (see *REQ-11*).

Figure 2 shows how the computation of the network results in interrelated network perspectives, which are accessed by the clients for network visualization, simulation or analytics in ⑨ (see *REQ-7,8,9*). For that, the inference result is translated into the BN model in ⑦ and compiled to a resource graph representation in ⑧ [12]. User updates are brought into the system through the client API and are visible (at least) for the authors (see *REQ-10*). In each of the steps the data provenance is updated to preserve the path to the origin of the data from the client queries to the source models (see *REQ-11*). Together with *REQ-12*, an end-to-end lineage from the original source data artifacts to the visualized instances of the computed network shall be possible. Through the separation of a storage for raw data (i.e., optimized for data write operations) and one for the BN (i.e., optimized for concurrent read access) *REQ-13* can be satisfied.

2.2 The BNS Models

The premise of NM is that all information required to compute the BN is available in the enterprises' landscapes. In fact, the information can be found

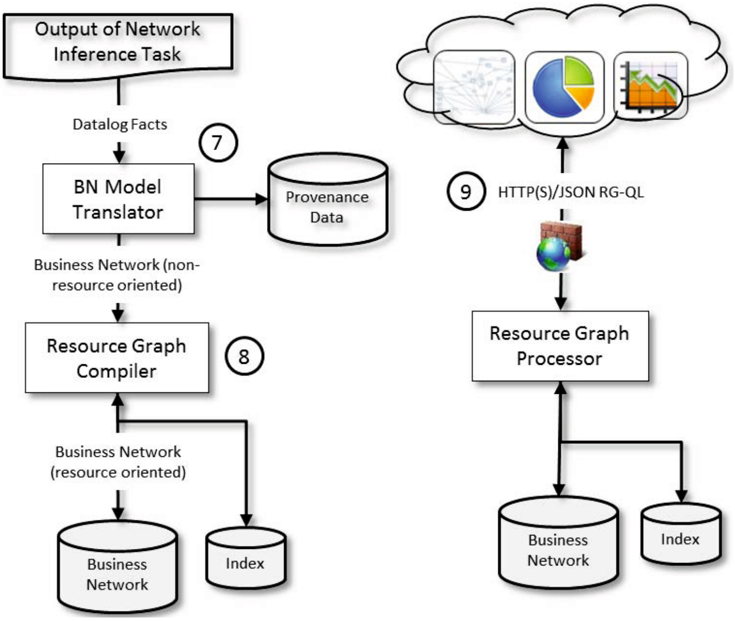


Fig. 2. System stack of BN translation, resource creation and client access

scattered across divers data sources, which come with different meta-data, formats, data semantics and quality (e.g., a SAP PI Business System [16] represents a node in the BN, later called participant or system. The information about its interfaces can be found in the middleware system, e.g., SAP PI, however its description and physical host it runs on is usually maintained in a system landscape directory, e.g., SAP SLD [17]). The network inference approach must not know about the domain-specificities, but should be more generally able to identify an entity and assign all its data fragments (see *REQ-2*). Hence, the BNS provides deployable components specific to the different domains, which can be configured to extract information from the data sources, pre-analyzes and transforms it into the inference model, defined in [15] (see Figure 1 ①). Since in our approach the inference programs are written as Datalog rules [18], the inference model is represented as Datalog facts. Figure 3 (right) shows the node part of the inference model, whose basic entities are *system* and *host*. Each entity in the inference model (e.g., *node*) holds a reference to its origin (i.e., meta-data about the data source and original object instance).

The distributed, domain-specific analysis decentralizes the information discovery process and guarantees that no domain logic has to be encoded in the inference programs (see *REQ-2*). The decentralized discovery components collect the transformed information and push them to the inference system at a configurable schedule. With this operation, the current state of the system is moved to the inference system, which stores the information in the local knowledge base as raw data. Through that, the inference programs only rely on local raw data,

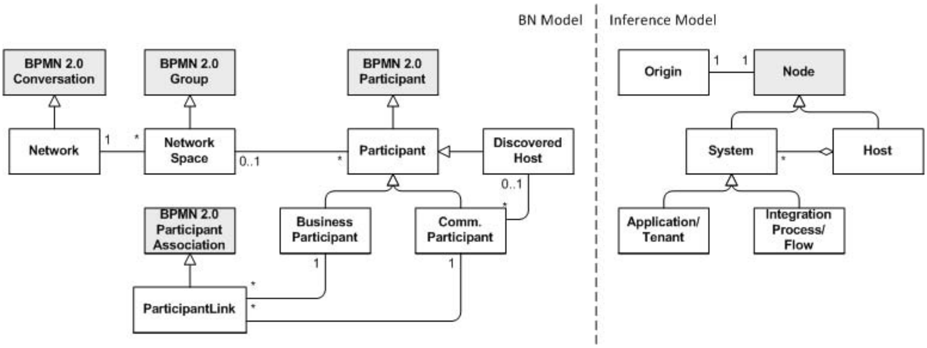


Fig. 3. Excerpts from the inference model (right) and BN model (left) showing only parts of the node definition

which ensures more efficient processing and makes it less dependent on network communication. However, the attached *origin* information keeps the link to the original records in the respective data sources (i.e., needed for provenance and continuous loading from the same source).

In Datalog, the raw data represents the extensible database (EDB) relations, which are evaluated by intensional database (IDB) rules (i.e., the inference programs). The result of the evaluation is the BN, which is already represented in a computer readable, visualizable standard as network-centric BPMN [10]. Figure 3 (left) shows the node part of the BN model, derived from the BPMN 2.0 collaboration-conversation diagram. The *Participant* corresponds to the *system* from the inference model. In addition, the inference programs compute *ParticipantLink*, which represent semantic contextualization between participants from different network perspectives (see see *REQ-7*). The business process and integration perspectives are built-in and defined as *NetworkSpaces*, which are part of the *Network* (i.e., BN; see *REQ-7*).

2.3 Declarative Network Mining and Extensibility

The choice for two models throughout the whole architecture (i.e., inference model and BN model) helps to separate the distributed discovery and inference from the (remote) access for search, query and traversal (see ⑥, ⑨). Furthermore this allows for a possibly different model evolution between the two models (e.g., new fields, entities added to the BN do not necessarily need to be propagated to the inference model). That means, only information from the automatic discovery has to be added in the inference model and programs. More concrete, an end-to-end model extension would require the ① inference model and ② its conformance checks, ⑦ the BN model, ⑥ the inference programs, and ⑧ the indices for query and traversal to change (depicted in Figures 1 or 2). Clearly, this is no task that should be done manually in the source code. Rather a configurable, declarative, model-centric approach is preferable (see *REQ-4*).

Figures 1 or 2 show the end-to-end declaration and configuration approach chosen for the BNS. The inference model provides the specification for the compilation and configuration of the conformance checks (details on the DSL are in [14] and the Datalog programs for the inference task are in [15]). While the conformance checks can be generated and configured without additional manual adaptation (see *REQ-6*), the inference programs can only partially be generated, since they represent domain expert knowledge (e.g., business process, integration), which cannot be derived from the inference model yet. However, this is a natural extension point, which allows domain experts to write their own inference programs to tweak the BN (see *REQ-4*). The BN model has to be adjusted before to allow the Datalog program to reference the modified entities. The resource graph representation and index structures for search, query and traversal of the BN adapt to changes in the BN model through a generic model interpretation (see requirement *REQ-2*). The remaining major, manual effort are the mapping of the domain-specific artifacts to the modified inference model and the adaptation of the source code that discovers the information in the enterprise landscape. Since these tasks still require significant expert knowledge and vary between enterprise landscapes, the automation is left for further research.

3 Query and Update Processing Overview

The data management capabilities of the BNS can be determined by the data flow. The data flow through the BNS covers client query and the update processing from the data sources. Figure 1 shows the update processing through the BN data discovery from the enterprise system landscapes ①. The process of discovering the source domain models can be scheduled and thus the raw data (represented as Datalog facts) does not necessarily be persisted before uploading the BNS. During the upload of the data it is checked for conformance with the inference model (see ②) by an automata-based runtime, compiled and configured from validation programs. The validation programs are declaratively developed on top of the inference model. If the check is successful, the data is stored in the knowledge base as raw data (see ③). More precisely, the conform raw data is stored, while keeping potential duplicate information. However, the unique identifiers from the source systems may not be unique in the inference system. To avoid "collisions" in case identifier occur more than once across different sources, the records are stored with a composed key containing their locally unique identifier and their origin. Keeping this in mind, records from the same origin with the same identifier are updated (i.e. remain as one record), while all other cases lead to different records (i.e. same origin, different keys; same key, different origin). That means, if a record is pushed to the inference system, which already contains a record with the same key and origin, the records are merged. In case of records without any primary key, the identity of the information cannot be determined. Hence a hash function is calculated over the record values, which leads to new records whenever the information is loaded. It then is the task of the inference programs to identify equivalence between the records and chose a meaningful surrogate. There are cases, in which the record has more than one identifiers. These are simply represented as combined primary key. The lineage tracing for the black-box transformation leverages the unique keys within the source combined with the origin information, which directs to the correct source. In this way, an anchor to the sources is created, which however lies in the storage close to the inference system.

The stored raw data represents the EDB for the inference programs. These programs are executed as inference tasks (see ④), which are analyzed with respect to partitioning and distribution over processing units in the system according to the entity dependency in the inference model and the programs themselves (see ⑤). For instance, the semantic relations $same\_system(sys_1, sys_2)$ and $runs\_on(sys, host)$, which require the sets of logical systems and hosts with their origin facts, can be scheduled as separate jobs, partitioned and distributed among different processing units. The corresponding optimized plans would look like $job_{any}(program_{any}, facts_{any})$, where $program_{any}$ is the set of inference programs to determine the semantic references, and $facts_{same} := system, \sigma_{system}(origin)$ and $facts_{runs} := system, host, \sigma_{system,host}(origin)$ are the partitioned sets of the EDB. The selection σ on $system$ and $host$ ensures that only the corresponding origins are taken into account.

One of the tasks of the optimized inference programs is to find equivalences between the same entities of the BN model (see ⑥). Due to the nature of equivalence classes, the most common operations are data copies (i.e., direct lineage tracing) and data aggregations. Since the identifiers in the BN become immediately visible to the applications, new keys are generated and a key mapping is maintained (see ⑦). The more difficult case is the aggregation of information, for which at least three variants are possible: (1) perform a "destroying merge" that identifies a leading set of data (the leading object or surrogate) and enrich the record by missing information from equivalent objects (e.g. add description to the leading object) and update the surrogate by any change from the sources, (2) perform a "preserving merge", which keeps all equivalent records and identifies a surrogate filled up with missing information (similar to (1)), while remembering from which object the information in the surrogate came from, or (3) do not aggregate at all, but make the equivalence information public for the applications and let them handle the merge. Although it comes with the highest data management efforts, the BNS supports the "information preserving", surrogate approach (2), which fulfills *REQ-11* for the BNS best. With that, the lineage tracing down to the sources (i.e., for operations on discovered records) and the steady integration of new sources and updates is granted.

The result of the translation from Datalog facts to the BN is then compiled to a resource graph structure (see ⑧), which automatically layouts the access paths and index structures for the client access and stores the "resource-oriented" BN for scalable access from the clients. The query processing capabilities of this system cover "get-everything" and selective queries, full-text search and traversal on the linked data in the BN. Hereby, the requests are formulated according to the business network protocol as RGQL (Resource Graph Query Language) [12].

For instance, the following RGQL specifies a *keyword search* with search term and result set restriction to *Host* (see ⑨)

`http://localhost/search?query=term&type=Host&...`

and *field*, as field specific search criteria:

`http://localhost/search?location=Oxford.`

In the same way, the result set can be defined to return any information in the BN by traversing the network, e.g., from a specific participant *system1*, e.g., `http://localhost/SYSTEM1/?show=meta,location,host.name`, which returns *location* information of the participant itself and the *name* of the linked host the participant runs on. Simple Friend of a Friend (FoaF) queries returning, e.g., the hosts of all neighbors of the participant are equally straight forward, e.g., `http://localhost/SYSTEM1/neighbors/host/`. Due to the decoupling of the data query and traversal components from the network inference and model-centric index generation, all requests are processed within short time even on larger networks (see [12] for performance numbers).

4 Related Work

For the overall system approach, related work is conducted in the area of Process Mining (PM) [1], which sits between computational intelligence and data mining. It has similar requirements for data discovery, conformance and enhancement with respect to NM [11], but does not work with network models and inference. PM exclusively strives to derive BPM models from process logs. Hence PM complements BNM in the area of business process discovery.

Gaining insight into the network of physical and virtual nodes within enterprises is only addressed by the *Host* entity in NIM, since it is not primarily relevant for visualizing and operating integration networks. This domain is mainly addressed by the IT service management [8] and virtualization community [5], which could be considered when introducing physical entities to our meta-model.

The linked (web) data research, shares similar approaches and methodologies, which have so far neglected linked data within enterprises and mainly focused on RDF-based approaches [3,4]. Applications of Datalog in the area of linked data [9,2] and semantic web [7] show that it is used in the inference domain, however not used for network inference.

5 Discussion and Future Work

In this work, we present insights into a reference implementation of the data management within a Business Network System based on the theory on Business Networks [11] and Business Network Management [13]. For that, we combined our work on conformance checking [14], business network inference [15] and a client API [12] into an emergent, enterprise-ready architecture. The architecture consitutes a holistic network data management platform, reaching from the information retrieval, network mining and inference, up to the BN.

The data provenance topic for the re-deployment of user information to the sources requires further as well as the topics of declarative, automatic information retrieval and inference program generation.

References

1. van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes (2011)
2. Abiteboul, S.: Distributed data management on the web. In: Datalog 2.0 Workshop, Oxford (2010)
3. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data – The Story so Far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
4. Bizer, C.: The Emerging Web of Linked Data. IEEE Intelligent Systems 24(5), 87–92 (2009)
5. Chowdhury, N.M.M.K., Boutaba, R.: Network virtualization: state of the art and research challenges. IEEE Communications Magazine (2009)
6. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Longman, Amsterdam (2003)
7. Motik, B.: Using Datalog on the Semantic Web. In: Datalog 2.0 Workshop, Oxford (2010)
8. O'Neill, P., et al.: Topic Overview – IT Service Management. Technical Report, Forrester Research (2006)
9. Polleres, A.: Using Datalog for Rule-Based Reasoning over Web Data: Challenges and Next Steps. In: Datalog 2.0 Workshop, Oxford (2010)
10. Ritter, D., Ackermann, J., Bhatt, A., Hoffmann, F.O.: Building a business graph system and network integration model based on BPMN. In: Dijkman, R., Hofstetter, J., Koehler, J. (eds.) BPMN 2011. LNBIP, vol. 95, pp. 154–159. Springer, Heidelberg (2011)
11. Ritter, D.: From Network Mining to Large Scale Business Networks. In: International Workshop on Large Scale Network Analysis (WWW Companion), Lyon (2012)
12. Ritter, D.: The Business Graph Protocol. In: The 18th International Conference on Information and Software Technologies (ICIST), Kaunas (2012)
13. Ritter, D.: Towards a Business Network Management. In: The 6th International Conference on Research and Practical Issues of Enterprise Information Systems, Confenis, Ghent (2012)
14. Ritter, D., Rupprich, S.: A Data Quality Approach to Conformance Checks for Business Network Models. In: The 6th IEEE International Conference on Semantic Computing (ICSC), Palermo (2012)
15. Ritter, D.: A Logic Programming Approach for the Reconstruction of Integration Networks. In: The 26th Workshop on Logic Programming (WLP), Bonn (2012)
16. SAP Process Integration (2012), `http://www.sap.com/germany/plattform/netweaver/components/pi/index.epx`
17. SAP System Landscape Directory (2012), `http://scn.sap.com/docs/DOC-8042`
18. Ullman, J.D.: Principles of Database and Knowledge-Base Systems, vol. I. Computer Science Press (1988)

Assessing the Completeness of Geographical Data

Simon Razniewski and Werner Nutt

Free University of Bozen-Bolzano
{razniewski,nutt}@inf.unibz.it

Abstract. Geographical databases are often incomplete, especially when built up incrementally and by volunteers. A prominent example is OpenStreetMap. Often such databases contain also metadata saying that certain features are completely captured for certain areas. We show how to use such metadata to analyse in which areas queries return a complete answer. Such "completeness areas" can be computed via standard spatial operations. Still larger completeness areas can be derived if not only metadata but also the actual content of the database is taken into account. Finally, we discuss which challenges arise if one wants to practically utilize the completeness metadata in OpenStreetMap.

1 Introduction

Storage and querying of geographic information has always been important. Recently, due to the increased availability of GPS devices, volunteered geographical information systems, in particular OpenStreetMap, have quickly evolved. Ongoing open public data initiatives that allow to integrate government data also contribute. The level of detail of OpenStreetMap is generally significantly higher than that of commercial solutions such as Google Maps or Bing Maps, while its accuracy and completeness are comparable.

OpenStreetMap (OSM) allows to collect information about the world in remarkable detail. This, together with the fact that the data is collected in a voluntary, possibly not systematic manner, brings up the question of the completeness of the OSM data. When using OSM, it is desirable also to get metadata about the completeness of the presented data, in order to properly understand its usefulness.

Judging completeness by comparing with other data is only possible, if more reliable data exists, which is generally not the case. Therefore, completeness can best be assessed by metadata about the completeness of the data, that is produced in parallel to the base data, and that can be compiled and shown to users. In geographical database systems it is very common to collect metadata, as widespread standards such as the FGDC metadata standard show. However, little is known about how query answers can be annotated with completeness information.

As an example, consider that a tourist wants to find hotels in some town that are no further than 500 meters away from a spa. Assume, that, as shown in Figure 1, the data about hotels and spas is only complete in parts of the map. Then, the query answer is only complete in the intersection of the areas where hotels are complete and a zone 500 meters inside the area where spas are complete (green in the figure), because outside, either hotels or spas within 500 meters from a hotel could be missing from the database, thus leading to missing query results.

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 228–237, 2013.

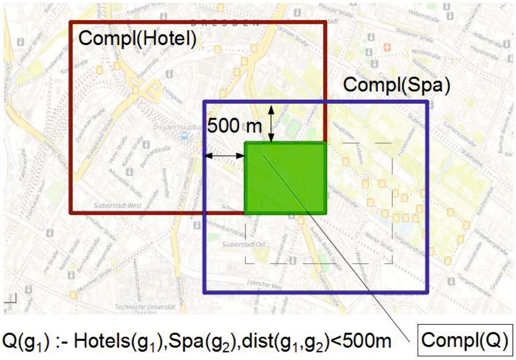


Fig. 1. Spatial query completeness analysis example. For a more complex one, see Figure 3.

Our contribution in this paper is a general solution for reasoning about the completeness of spatial data. In particular, we show that metadata can allow elaborate conclusions about query completeness, when one takes into account the data actually stored in the database. We also show that metadata about completeness is already present to a limited extent for OSM, and discuss practical challenges regarding acquisition and usage of completeness metadata in the OSM project.

The structure of this paper is as follows: In Section 2, we discuss spatial database systems, online map services, geographical data completeness and OpenStreetMap. In Section 3, we give necessary formalizations to discuss the problem of reasoning about geographical data completeness in Section 4. In Section 5, we discuss practical issues regarding the applicability of our solution to OpenStreetMap.

2 Background

2.1 Spatial Databases Systems and Online Map Services

To facilitate storage and retrieval, geographic data is usually stored in spatial databases. According to [1], spatial databases have three distinctive features. First, they are database systems, thus classical relational/tree-shaped data can be stored in them and retrieved via standard database query languages. Second, they offer spatial data types, which are essential to describe spatial objects. Third, they efficiently support spatial data types via spatial indexes and spatial joins.

Online map services usually provide graphical access to spatial databases and provide services for routing and address finding. There are several online map services available, some of the most popular ones being Google Maps, Bing Maps, MapQuest and OpenStreetMap. With the exception of OSM, the data underlying those services is not freely accessible. The most common uses of those services are routing ("Best path from A to B?"), address retrieval ("Where is 2nd street?") and business retrieval ("Hotels in Miami"). While the query capabilities of most online map services are currently still limited (one can usually only search for strings and select categories), spatial databases generally allow much more complex queries.

Example 1. Tourists could be interested in finding those hotels that are less than 500 meters from a spa and 1 kilometer from the city center. Real estate agents could be interested in properties that are larger than 1000 square meters and not more than 5 kilometers from the next town with a school and a supermarket. Evacuation planners might want to know which public facilities (schools, retirement homes, kindergartens, etc.) are within a certain range around a chemical industry complex.

2.2 Geographical Data Completeness

In addition to precision and accuracy, knowledge about completeness is essential when using spatial data [2]. Completeness describes the extent to which features in the real world that are of interest are also present in the database. Completeness can highly vary for different features. If metadata about completeness is present, it is attractive to visualize it on maps [3]. Completeness is especially a challenge when (1) databases are to capture continuously the current state (as opposed to a database that stores a map for a fixed date) because new features can appear, (2) databases are built up incrementally and are accessible during build-up (as it is the case for OSM) and (3) the level of detail that can be stored in the database is high (as it is easier to be complete for all highways in a state than for all post boxes).

2.3 OpenStreetMap

OpenStreetMap is a wiki-style project for building a map of the world. Contributors to the project are volunteers that commonly use GPS devices to track paths and features or use aerial pictures to identify them. In contrast to most commercial products, the project's data is freely available and map data can be reused in other applications and visualized for different specific needs. In most areas of the world, OSM provides significantly more information than its commercial competitors.[1] The level of detail of features that can be stored is remarkable, as for example for buildings, entrances and building heights can be stored, for vending machines the kind of good they sell or for waste baskets the kind of waste they accept.

There have been some attempts at formalizing [4] and assessing the quality of OSM [5,6]. The latter showed that the road map coverage of England is quite good, but it is clear that, because of the level of detail that can be stored, many non-core features are currently far from being complete.

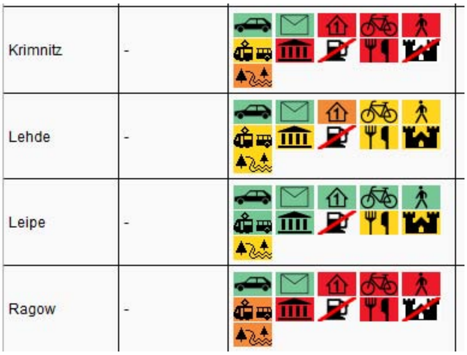
Assessment of completeness is not only important for consumers but also for the producers of the data (mappers), because they are interested to know where to direct their work. Some OSM mappers therefore introduced the idea of storing information about completeness of different features in tables on the OSM wiki [7]. An example of such a table is shown in Figure 2.

3 Formalization

3.1 Spatial Databases

A *geometry* is a datatype used to represent the location and extent of a spatial object, for example as a point or a polygon.

[1] For a graphical comparison tool, see for example `http://tools.geofabrik.de/mc/`.



Krimnitz	-	
Lehde	-	
Leipe	-	
Ragow	-	

Fig. 2. Completeness information for different districts of Lübbenau, Germany. Each symbol stands for a set of features, the colors for its completeness level. Taken from [8].

We assume that a *spatial database* consists of a set of relations, where each relation has besides other attributes exactly one attribute g of type geometry.

We consider a special class of queries, which we call star-shaped queries, that allow only joins over the distance-less-than relation with the output relation. Formally, a *star-shaped query* is written as

$$\begin{aligned} Q(g_0) :- R_0(\bar{x}_0, g_0), M_0, R_1(\bar{x}_1, g_1), M_1, \mathrm{dist}(g_0, g_1) < c_1, \ldots, \\ R_n(\bar{x}_n, g_n), M_n, \mathrm{dist}(g_0, g_n) < c_n, \end{aligned} \tag{1}$$

where each M_i is a set of comparisons of the arguments of R_i with constants.

Example 2. Consider again the query asking for hotels that are closer than 500 meters to a spa and closer than 1 km to the city center. This can be written as

$$\begin{aligned} Q_{\text{hotels}}(g_0) :- \ \mathrm{Hotel}(n, r, g_0), \mathrm{Spa}(g_1), \mathrm{dist}(g_0, g_1) < 500, \\ \mathrm{Center}(g_2), \mathrm{dist}(g_0, g_2) < 1000, \end{aligned}$$

if we assume a ternary relation Hotel with attributes name, rating, and geometry and unary relations for spas and centers. The query is star-shaped, because both spa and center are joined with hotel, which is the relation providing the output. The other queries in Example 1 are star-shaped as well.

3.2 Completeness

Real-world databases are often incomplete. This was formalized by Motro [9] such that incomplete databases are actually pairs (D^i, D^a), where the *available* (real) database D^a contains only a subset of the facts that hold in the *ideal* database D^i, which represents the complete information about the world.

Example 3. Consider that in the real world, there are a Hilton (4 stars), a Sheraton (5 stars) and a Best Western (3 stars) hotel in a town, but in the database, the information about the Best Western is missing. Ideal and available database for that scenario are shown in Table 1.

Table 1. Example of an ideal and available database

Ideal database — Relation Hotel		
name	rating (stars)	geometry
Hilton	4	P(46.61, 12.30)
Sheraton	5	P(46.62, 12.30)
Best Western	3	P(46.64, 12.26)

Available database — Relation Hotel		
name	rating (stars)	geometry
Hilton	4	P(46.61, 12.30)
Sheration	5	P(46.62, 12.30)

To express partial completeness, Levy introduced a format for metadata about database completeness, which he called local completeness statements [10]. We extend those to feature completeness statements. Formally, a *feature completeness statement* F is written as Compl($R(\bar{x}, g)$; M; A) and consists of a relation name R, a set M of comparisons of the attributes of R with constants, and an area A. The feature completeness statement holds over a pair of an ideal and an available database (D^i, D^a), if its associated query $Q_F(g) :\!- R(\bar{x}, g), M$ returns the same answers over both databases in the area A, that is, if $Q_F(D^i) \cap A = Q_F(D^a) \cap A$.

Example 4. The database in Table 1 is complete for all hotels with more than 3 stars on the full map, that is, it satisfies the statement F_1 = Compl(Hotel(n, r, g); $\{r > 3\}$; *FULL_MAP*). Furthermore, it is complete for all hotels in areas south of 46.63 latitude, that is, it satisfies F_2 = Compl(Hotel(n, r, g); $\emptyset$; *RECTANGLE*(0, 0 46.63, 20)).

Let $\mathcal{F}$ be a set of feature completeness statements $\mathcal{F}$ and Q be a star-shaped query. We define the completeness area CA of Q wrt $\mathcal{F}$ as the set of points p such that for all pairs of an ideal and an available database (D^i, D^a) that satisfy $\mathcal{F}$, it holds that $p \in Q(D^i)$ implies $p \in Q(D^a)$. In other words, Q will not miss p as an answer over the available database if it would return p over the ideal database. Thus, CA is the largest area such that $Q(D^i) \cap CA = Q(D^a) \cap CA$ for all (D^i, D^a) satisfying $\mathcal{F}$.

Example 5. Consider the query $Q_{\text{FiveStars}}(g) :\!- \text{Hotel}(n, r, g), r = 5$ that asks for all hotels with five stars. Assuming only the feature completeness statement F_1 from above holds, CA is FULL_MAP. Assuming, however, only the statement F_2 holds, CA is RECTANGLE(0, 0 46.63, 20).

4 Query Completeness Analysis

Given a query Q and a set of FC statements $\mathcal{F}$, the completeness analysis problem is to determine CA. We first analyse the general problem, then show that larger completeness areas can be derived, if the available database is taken into account. For simplicity, we assume in the following that the geometries of all features are points, and consider only star-shaped queries.

4.1 Analysis without the Database Instance

We first consider completeness analysis problem for *simple* queries, which are of the form $Q(g) :\!- R(\bar{x}, g), M$.

Let $\mathcal{F}$ be a set of FC statements and R be a relation. Then we denote by $\mathcal{F}^R$ the subset of $\mathcal{F}$ that consists of those statements that talk about R. Suppose $F_1, \ldots, F_m$ is an enumeration of the statements in $\mathcal{F}^R$. Then each F_j in $\mathcal{F}^R$ is of the form Compl($R(\bar{x}, g)$; $M_j; A_j$), where M_j is a set of comparisons over the variables in $\bar{x}$ and A_j is an area.

Proposition 1. *Let $\mathcal{F}$ be a set of FC statements and $Q(g) :\!- R(\bar{x}, g), M$ be a simple query. Then CA, the completeness area of Q wrt $\mathcal{F}$, satisfies*

$$CA = \bigcup \{ \bigcap_{F_j \in \mathcal{F}_0} A_j \mid \mathcal{F}_0 \subseteq \mathcal{F}^R \text{ and } M \models \bigvee_{F_j \in \mathcal{F}_0} M_j \}.$$

The preceding proposition says that CA is a union of areas $\bigcap_{F_j \in \mathcal{F}_0} A_j$, which are obtained as follows. One chooses a subset $\mathcal{F}_0$ of statements F_j, such that the query condition M entails the disjunction of the conditions M_j of the F_j. Intuitively, this means that the M_j cover all possible ways in which M can be satisfied. Then one intersects the areas A_j of the statements in $\mathcal{F}_0$. Of course, it suffices to take only minimal subsets $\mathcal{F}_0$ of $\mathcal{F}^R$ (wrt set inclusion) whose comparisons are entailed by M.

We remark that the decision version of the problem to determine the CA for a simple query ("Given $\mathcal{F}$, Q and a point p, is p in CA?") is coNP-hard. This can be seen by an encoding the tautology problem for propositional logic. The problem is also in coNP because to show that a point is not in CA, it suffices to guess values for the attributes of R such that none of the conditions of the FC statements that hold in p are satisfied. The hardness however may be practically not very problematic, because (i) it is maybe rather uncommon that FC statements only together imply completeness, because for that, they must complement each other, while it is more realistic, that they only linearily extend each other, and (ii) because geometries of FC statements may seldomly be overlapping, since they are expected to be given for administrative units.

To formulate a characterization of the completeness sets of arbitrary star-shaped queries, we need some additional notation. For an area A and a number $c > 0$, we denote with $shrink(A, c)$ the set of points p in A such that the distance between p and the complement of A is at least c. Intuitively, these are points that are lying deep in A.

For an arbitrary star-shaped query Q as in Equation (1) we introduce $n + 1$ simple queries $Q_0, \ldots, Q_n$, defined as $Q_i(g_i) :\!- R_i(\bar{x}_i), M_i$ for $i = 1, \ldots, n$, which we call the component queries of Q.

Theorem 1. *Let $\mathcal{F}$ be a set of FC statements, $Q(g_0)$ a query as in Equation (1), and $Q_0, \ldots, Q_n$ the component queries of Q. Then CA, the completeness area of Q wrt $\mathcal{F}$, satisfies*

$$CA = CA_0 \cap shrink(CA_1, c_1) \cap \ldots \cap shrink(CA_n, c_n),$$

where CA_i is the completeness area of Q_i.

Example 6. See again Figure 1. There, hotels are complete within the brown rectangle and spas within the blue. The area $shrink$(Spa, 500) is indicated by the dashed line. Completeness of the query only holds inside the green area, because for any point outside the brown rectangle, there could be a hotel missing in the database that has a spa nearby, and for any point outside $shrink$(Spa, 500) there could be a spa missing outside the blue rectangle, that indicates that a hotel has the property of having a spa within 500 meters.

Algorithms to compute CA_i and $CA(Q, \mathcal{F})$ are given below:

```
CompleteArea(Feature R, Conditions M, Set of FC statements F)
  Area = empty
  for each subset S of F:
    if statements in S imply completeness of R,M
    then Area = Area.union(intersection of geometries in S)
  return Area
```

```
CalcQueryComplW/oInstance(Query Q, Set of FC statements F)
  Area = CompleteArea($R_0$, $M_0$, F)
  for each pair $R_i$, $M_i$ i>0 in Q:
    AtomArea = ShrinkArea(CompleteArea($R_i$,$M_i$,F),$c_i$)
    Area = Area.intersect(AtomArea)
  return Area
```

Listing 1.1. Algorithms to compute the complete area for an atom and for a query

4.2 Analysis with the Database Instance

When analysing query completeness, one can also take into account the actual state of the database. The problem is then, given a query Q, a set of FC statements $\mathcal{F}$ and an available database instance D^a, to find the largest area CA where it holds for all ideal databases D^i, that, whenever (D^i, D^a) satisfies $\mathcal{F}$, then $Q(D^i) \cap CA = Q(D^a) \cap CA$. Taking into account the available database, more completeness can be derived.

Example 7. Consider Figure 3. The completeness statements are the same as in Figure 1, however now, there are also the positions of the hotels h_1, h_2, h_3 and spas s_1 and s_2 shown, as they are stored in D^a. The area CA is now significantly larger for two reasons:

1. All points within Compl(Hotel), where no hotel is located that could possibly be an answer (i.e., all points except h_2), are now added to CA. The reason is that for all those points it holds that there cannot be any hotel missing that satisfies the query condition, because hotels are complete in that point.
2. All points within *shrink*(Compl(Spa), 500m) where no spa is within a distance of 500m are added, because for those points, although hotels could possibly be missing from the database, none can be missing that satisfies the query condition, because there are no spas around within 500 meters and spas are complete within 500 meters.

To formalize our observations, we first introduce two definitions. Consider a query Q as in Equation (1), a set of FC statements $\mathcal{F}$ and an available database D^a.

We say that a point p is a *potential answer* to Q, if (i) there is an atom $R(\bar{t}, p)$ in $R(D^a)$, (ii) p is not in $Q(D^a)$ and (iii) there exists an ideal database D^i with $(D^i, D^a) \models \mathcal{F}$ such that $p \in Q(D^i)$. We denote the set of all potential answers of Q by $Pot(Q)$. The potential answers can be calculated by removing from $\pi_g(R)$ all those points that are a certain answer (i.e., are in $Q(D^a)$ and that are an impossible answer (i.e., cannot be in $Q(D^i)$

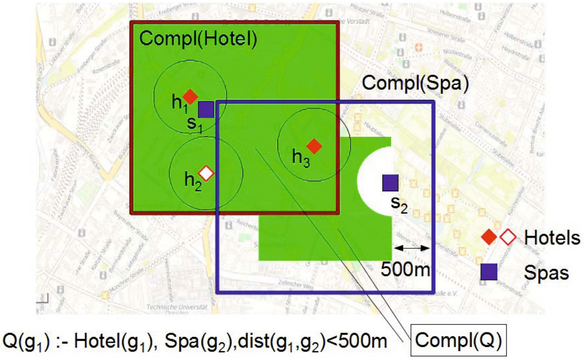


Fig. 3. Completeness analysis when taking into account the database instance. Observe that the point h_2 does not belong to the completeness area.

because some join partner A_i is not in the range c_i in D^a and the atom is also complete in the area buffer(p, c_i), which is the set of points with a distance of at most c_i from p.

For a pair $G_i = R_i(\bar{x}), M_i$ in Q, we define the area $ComplOutOfRange_{\mathcal{F},D^a}(G_i)$ as $shrink(CA_{A_i}, c_i) \setminus \bigcup\{buffer(p, c_i) \mid p \in Q(g) :\!- A_i(\bar{x}, g), M_i\}$.

Now, we can compute the largest area in which a query Q is complete wrt. F and D^a by taking the union of the area where G_0 is complete with the areas where each G_i is complete and out of reach in the distance c_i, and finally removing the points where potential answers are located:

Theorem 2. *Let Q be a star-shaped query with n atoms, F be a set of FC statements and D^a be an available database. Then*

$$CA(Q, F, D^a) = CA_{A_0} \cup (\bigcup_{i=1...n} ComplOutOfRange_{\mathcal{F},D^a}(G_i)) \setminus \pi_g(Pot(Q)).$$

5 Practical Issues in OpenStreetMap

In the OpenStreetMap-wiki, a template for tables that store completeness information exists (see Figure 2). The template introduces 6 levels of completeness (“Largely incomplete” to “Completeness verified by two mappers”) for 11 different features (such as roads or sights), and is used on approximately 1,100 Wiki pages (estimate based on number of pages that contain an image used in the table), which corresponds to 5% of all pages on the Wiki (21,989 content pages on 22.01.2013).

The table in Figure 2 expresses for example that the roads in Lehde are generally complete, the cycling paths largely and the house numbers partially.

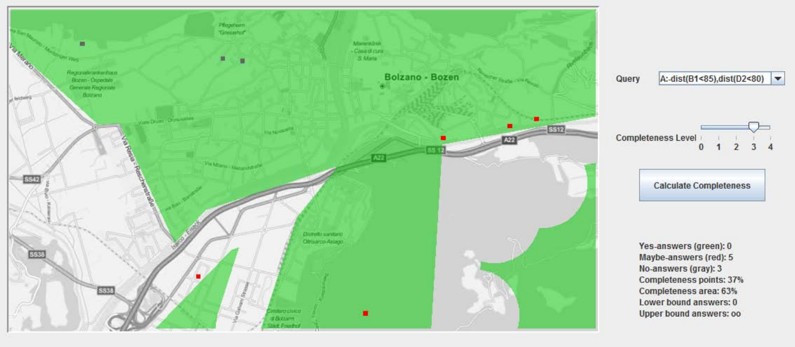
The completeness statements that can be expressed via those tables are of a simpler form than the ones discussed before, because there cannot be any comparisons. Also, there can be at most one completeness statement per relation per area, and the areas are disjoint, since they talk about different administrative units. Altogether, this makes the query completeness analysis computationally easy.

Practically challenging is especially the proper interpretation of those completeness levels: It is hard to say what “Largely complete” can actually mean, and whether that level of completeness is satisfactory for a user’s request. A possible solution would be to use percentages instead of informal descriptions for completeness statements, however the problem then is that giving correct numbers (“60% complete”) is only possible, if one has a good estimate of what 100% would be.

Another serious challenge is to get mappers to widely give those completeness statements. The current usage (5%) is clearly insufficient. A possible reason is that completeness statements introduce a kind of negative responsibility: One states, that there are no unmapped features in a certain kind of area. This is a much stronger statement than saying that a feature is present at some place, which is the usual implicit statement that mappers give when adding a feature to the map.

A third challenge is provided by changes in the real world: While during the build-up of a geographical database changes in the real world are less important, during longer runs, also changes becomes a problem, because features that can have been correct in the past can disappear (e.g. hotels can close), or new features can appear (e.g. new hotels can open). Thus, completeness statements would need periodic review, a possibly not very attractive activity (Wikipedia has a similar problem with its review system).

Minor technical challenges are that the completeness statements would need to be crawled from the OSM-wiki pages, that the district borders used in the Wiki do not always exist in the database and that the OSM data is not stored in relational format but in XML format.



Query A:-dist(B1<85),dist(D2<80)

Yes-answers (green): 0
Maybe-answers (red): 5
No-answers (gray): 3
Completeness points: 37%
Completeness area: 63%
Lower bound answers: 0
Upper bound answers: oo

Fig. 4. Screenshot of our demo program for reasoning about the completeness of OSM

6 Conclusion

We have shown that completeness is a major concern in geographical databases that are built up incrementally and by volunteers, and that that holds particularly for OpenStreetMap. We also showed that for analyzing database completeness, metadata is required, because completeness can normally not be analyzed by looking at the data itself.

We showed how metadata about feature completeness can be used to analyze query completeness, and that more completeness can be concluded when not only the metadata but also the database content is taken into account.

We discussed that major challenges for practical applicability are the willingness of mappers to give completeness statements and to review them regularly.

We also implemented a small test program that contains a fictive scenario for the town of Bolzano, which is available under [13]. A screenshot is also shown in Figure 4.

Future work will focus on implementation techniques for a tool that is built upon real OSM data and on discussing the applicability and usefulness of our proposal with the OSM community.

Acknowledgement. We are thankful to the user Bigbug21 for information about the OSM community.

References

1. Güting, R.H.: An introduction to spatial database systems. VLDB J 3(4), 357–399 (1994)
2. Shi, W., Fisher, P., Goodchild, M.: Spatial Data Quality. CRC (2002)
3. Wang, T., Wang, J.: Visualisation of spatial data quality for internet and mobile GIS applications. Journal of Spatial Science 49(1), 97–107 (2004)
4. Mooney, P., Corcoran, P., Winstanley, A.: Towards quality metrics for OpenStreetMap. In: Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 514–517. ACM (2010)
5. Haklay, M., Ellul, C.: Completeness in volunteered geographical information—the evolution of OpenStreetMap coverage in England (2008-2009). Journal of Spatial Information Science (2010)
6. Haklay, M.: How good is volunteered geographical information? A comparative study of OpenStreetMap and Ordnance Survey datasets. Environment and Planning. B, Planning & Design 37(4), 682 (2010)
7. OSM Wiki, First appearance of completeness statement in OSM (December 2007), http://wiki.openstreetmap.org/wiki/Talk:Landkreis_M%C3%BCnchen
8. OSM Wiki, Completeness statements in the OSM wiki for Luebbenau, http://wiki.openstreetmap.org/wiki/L%C3%BCbbenau
9. Motro, A.: Integrity = Validity + Completeness. ACM Transactions on Database Systems (TODS) 14(4), 480–502 (1989)
10. Levy, A.: Obtaining complete answers from incomplete databases. In: Proc. VLDB, pp. 402–412 (1996), http://portal.acm.org/citation.cfm?id=645922.673332
11. Razniewski, S., Nutt, W.: Completeness of queries over incomplete databases. Proc. VLDB Endow 4(11), 749–760 (2011)
12. Nutt, W., Razniewski, S.: Completeness of queries over sql databases. In: CIKM (2012)
13. Razniewski, S.: Completeness reasoning test program, http://www.inf.unibz.it/~srazniewski/geoCompl/

A Comprehensive Study of iDistance Partitioning Strategies for *k*NN Queries and High-Dimensional Data Indexing

Michael A. Schuh, Tim Wylie, Juan M. Banda, and Rafal A. Angryk

Montana State University, Bozeman, MT 59717-3880, USA
{michael.schuh,timothy.wylie,juan.banda,angryk}@cs.montana.edu

Abstract. Efficient database indexing and information retrieval tasks such as k-nearest neighbor (kNN) search still remain difficult challenges in large-scale and high-dimensional data. In this work, we perform the first comprehensive analysis of different partitioning strategies for the state-of-the-art high-dimensional indexing technique iDistance. This work greatly extends the discussion of why certain strategies work better than others over datasets of various distributions, dimensionality, and size. Through the use of novel partitioning strategies and extensive experimentation on real and synthetic datasets, our results establish an up-to-date iDistance benchmark for efficient kNN querying of large-scale and high-dimensional data and highlight the inherent difficulties associated with such tasks. We show that partitioning strategies can greatly affect the performance of iDistance and outline current best practices for using the indexing algorithm in modern application or comparative evaluation.

Keywords: iDistance, Large-scale, High-dimensional, Indexing, Retrieval, kNN.

1 Introduction

Modern database-oriented applications are filled with rich information composed of an ever-increasing amount of large-scale and high-dimensional data. While storing this data is becoming more routine, efficiently indexing and retrieving it is still a practical concern. A frequent and costly retrieval task on these databases is k-nearest neighbor (kNN) search, which returns the k most similar records to any given query record. While all database management systems (DBMS) are highly optimized for a few dimensions, the traditional indexing algorithms (*e.g.*, the B-tree and R-tree families) degrade quickly as the number of dimensions increase, and eventually a sequential (linear) scan of every single record in the database becomes the fastest retrieval method.

Many algorithms have been proposed in the past with limited success for truly high-dimensional indexing, and this general problem is commonly referred to as *the curse of dimensionality* [4]. Practitioners often mitigate these issues through dimensionality reduction techniques (manual and automated) before using multi-dimensional indexing methods, or even adding application logic to combine multiple independent indexes or requiring user involvement during search. However,

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 238–252, 2013.

modern applications are increasingly employing highly-dimensional techniques to effectively represent massive data, such as the highly popular 128-dimensional SIFT features [11] in Content-Based Image Retrieval (CBIR).

First published in 2001, iDistance [10,20] specifically addressed kNN queries in high-dimensional space and has since proven to be one of the most efficient and state-of-the-art high-dimensional indexing techniques available for exact kNN search. In recent years, iDistance has been used in a number of demanding applications, including large-scale image retrieval [21], video indexing [15], mobile computing [8], peer-to-peer systems [6], and surveillance system video retrieval [14]. Unfortunately, no works to date have focused on developing methods of best practice for these modern applications.

This work methodically analyzes partitioning strategies with the goal of increasing overall performance efficiency of indexing and retrieval determined by the total tree nodes accessed, candidate records returned, and the time taken to perform a query. These metrics are used to quantitatively establish best practices and provide benchmarks for the comparison of new methods. We introduce a new and open-source implementation of the original iDistance algorithm[1] including detailed documentation, examples, visualizations, and extensive test scripts. We also contribute research-supporting code for pre-processing datasets and post-processing results, as well as all published algorithmic improvements.

The motivations addressed in the original iDistance publications have only increased in importance because of the ubiquity of rich high-dimensional and large-scale data for information retrieval, such as multimedia databases and the mobile computing market which have exploded in popularity since the last publication in 2005. While there is little doubt the algorithm remains effective and competitive, a more thorough investigation into performance-affecting criteria is needed to provide a basis for general capabilities and best practices. Without this study, it can be difficult to effectively use iDistance in application and reliably compare it to new methods in future research.

The rest of the paper is organized as follows. Section 2 highlights background and related works, an overview of iDistance is presented in Section 3, and experiments and results are presented in Section 4. We follow with a brief discussion of key findings and best practices in Section 5, and we close with our conclusions and future work in Section 6.

2 Background and Related Works

The ability to efficiently index and retrieve data has become a silent backbone of modern society, and it defines the capabilities and limitations of practical data usage. While the one-dimensional B$^+$-tree [2] is foundational to the modern relational DBMS, most real-life data has many dimensions (attributes) that would be better indexed together than individually. Mathematics has long-studied the partitioning of multi-dimensional metric spaces, most notably Voronoi Diagrams and the related Delaunay triangulations [1], but these theoretical solutions can be

[1] Publicly available at: `http://code.google.com/p/idistance/`

too complex for application. R-trees [7] were developed with minimum bounding rectangles (MBRs) to build a hierarchical tree of successively smaller MBRs containing objects in a multi-dimensional space, and R*-trees [3] enhanced search efficiency by minimizing MBR overlap. However, these trees (and most derivations) quickly degrade in performance as the dimensions increase [5,13].

Recently, research has focused on creating indexing methods that define a one-way lossy mapping function from a multi-dimensional space to a one-dimensional space that can then be indexed efficiently in a standard B^{+}-tree. These lossy mappings require a filter-and-refine strategy to produce exact query results, where the one-dimensional index is used to quickly retrieve a subset of the data points as candidates (the filter step), and then each of these candidates is checked to be within the specified query region in the original multi-dimensional space (the refine step). Because checking the candidates in the actual dataspace is a costly task, the goal of the filter step is to return as few candidates as possible while retaining the exact results.

The Pyramid Technique [5] was one of the first prominent methods to effectively use this strategy by dividing up the d-dimensional space into $2d$ pyramids with the apexes meeting in the center of the dataspace. This was later extended by moving the apexes to better balance the data distribution equally across all pyramids [22]. For greater simplicity and flexibility, iMinMax(θ) [13,16] was developed with a global partitioning line θ that can be moved based on the data distribution to create more balanced partitions leading to more efficient retrieval. The simpler transformation function also aids in faster filter-step calculations for finding candidate sets. Both the Pyramid Technique and iMinMax(θ) were designed for range queries in a multi-dimensional space, and extending to high-dimensional kNN queries is not a trivial task.

It should also be briefly noted that many other works are focused on returning approximate nearest neighbors [9,18], but these are outside the scope of efficient exact kNN retrieval by iDistance presented in this paper.

3 iDistance

The basic concept of iDistance is to segment the dataspace into disjoint partitions, where all points in a specific partition are *indexed by their distance* ("iDistance") to the reference point of that partition. This results in a set of one-dimensional distance values, each related to one or more data points, for each partition that are all together indexed in a single standard B^{+}-tree. The algorithm was motivated by the ability to use arbitrary reference points to determine the (dis)similarity between any two data points in a metric space, allowing single dimensional ranking and indexing of data points no matter what the dimensionality of the original space [10]. The algorithm also contains several adjustable parameters and run-time options, making the overall complexity and performance highly dependent on the choices made by the user. Here we provide an overview of the algorithm and readers are encouraged to refer to the original works [10,20] for details that are beyond the scope of our investigation.

3.1 Building the Index

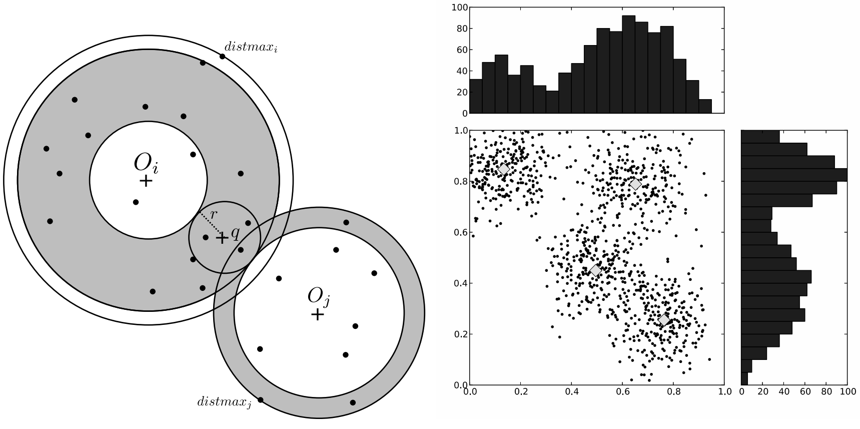
The most important algorithmic option for iDistance is the partitioning strategy. The original works presented two types of abstract partitioning strategies: *space-based*, which assumes no knowledge of the actual data, and *data-based*, which adjusts the size and location of partitions based on the data distribution [10,20]. For any strategy, every partition requires a representative reference point, and data points are assigned to the single closest partition in Euclidean distance.

A mapping scheme is required to create separation between the partitions in the underlying B$^+$-tree, ensuring any given index value represents a unique distance in exactly one partition. Given a partition P_i with reference point O_i, the index value y_p for a point p assigned to this partition is defined by Equation 1, where $dist()$ is any metric distance function, i is the partition index, and c is a constant multiplier for creating the partition separation. While constructing the index, each partition P_i records the distance of its farthest point as $distmax_i$.

$$y_p = i \times c + dist(O_i, p) \tag{1}$$

3.2 Querying the Index

The index should be built in such a way that the filter step returns the fewest possible candidate points without missing the true k-nearest neighbor points. Fewer candidates reduces the costly refinement step which must verify the true multi-dimensional distance of each candidate from the query point. Performing a query q with radius r consists of three basic steps: 1) determine the set of



(a) (b)

Fig. 1. (a) A query sphere q with radius r and the searched regions (shaded) in the two overlapping partitions P_i and P_j defined by their reference points O_i and O_j, and radii $distmax_i$ and $distmax_j$ respectively. (b) A scatter plot of a two dimensional dataset with four clusters, accompanied by each single dimensional histogram.

partitions to search, 2) calculate the search range for each partition in the set, and 3) retrieve the candidate points and refine by true distance.

Figure 1(a) shows an example query point q with radius r contained completely within partition P_i and intersecting partition P_j, as well as the shaded ranges of each partition that need to be searched. For each partition P_i and its $distmax_i$, the query sphere (q, r) overlaps the partition if the distance from the edge of the query sphere to the reference point O_i is less than $distmax_i$, as defined in Equation 2. There are two possible cases of overlap: 1) q resides within P_i, or 2) q is outside of P_i, but the query sphere intersects it. In the first case, the partition needs to be searched both inward and outward from the query point over the range $(q \pm r)$, whereas an intersected partition is only searched inward from the edge of the partition to the farthest point of intersection. Equation 3 combines both overlap cases into a single search range for each partition.

$$dist(O_i, q) - r \leq distmax_i \tag{2}$$

$$[dist(O_i, q) - r, MIN(dist(O_i, q) + r, distmax_i)] \tag{3}$$

3.3 Partition Strategies

Space-based Strategies. The only space-based methods presented in detail in previous works [10,20] were *Center of Hyperplane* and *External Point*, which we refer to in this work as *Half-Points* (HP) and *Half-Points-Outside* (HPO), respectively. The HP method mimics the Pyramid-Technique [5] by placing reference points at the center of each dimensional edge of the data space with $2d$ partitions in d dimensions. The HPO method creates the same reference points, but then moves them outside of the dataspace by a preset distance to reduce the overlap volume between partitions. For example, in a 2D space such as Figure 1(b), HP would result in four partitions, based on reference points: $(0.0, 0.5)$, $(0.5, 0.0)$, $(1.0, 0.5)$, and $(0.5, 1.0)$, and HPO-10 (movement of 10.0) would result in reference points: $(-10.0, 0.5)$, $(0.5, -10.0)$, $(11.0, 0.5)$, and $(0.5, 11.0)$ respectively. Here we also introduce random reference point selection (RAND) to create any number of partitions located randomly in the dataspace. While this is a trivial strategy, it has not been shown before and greatly helps compare other strategies by providing a naïve benchmark.

Data-based Strategies. The primary benefit of data-based methods is their adaptability to data distributions, which greatly increases retrieval performance in real-world settings. Two methods were originally introduced: *center of cluster* and *edge of cluster*, but only the *center of cluster* method was actually presented in published results [10,20], which used algorithmically derived cluster centers as reference points to create cluster-based partitions in the dataspace.

Approximate cluster centers can be found through a variety of popular clustering algorithms, such as k-Means [12], BIRCH [23], *etc.*, and the original authors recommend (without explicit rationale) to use $2d$ as the number of partitions (clusters). They believed using the edges of clusters is intuitively more promising as it should reduce partition overlap (decreasing node accesses) and reduce the number of equi-distant points from any given reference point (decreasing candidates). Unfortunately, they leave us with only implementation suggestions, such as "points on hyperplanes, data space corners, data points at one side of a cluster and away from other clusters, and so on" [10], but many of these methods are infeasible in high dimensions and were never presented.

4 Experiments and Results

We propose new iDistance partitioning strategies and methodically determine the effectiveness of various strategies over a wide range of dataset characteristics that lead to generalized conclusions about when and how to apply certain strategies (if at all). This not only depends on the dataset size and dimensionality, but also on additional knowledge possibly available, such as data distributions and clusters. We highlight these variabilities over extensive experiments that not only validate the results (and independent/unbiased reproducibility) of original research [10,20], but also greatly extend the analyses through novel strategies and specially designed dataspaces.

Every run of our implementation of iDistance reports a set of statistics describing the index and query performance of that run. As an attempt to remove machine-dependent statistics, we use the number of B^+-tree nodes instead of page accesses when reporting query results and tree size. Tracking nodes accessed is much easier within the algorithm and across heterogeneous systems, and is still directly related to page accesses through the given machine's page size and B^+-tree leaf size. We primarily highlight three statistics from tested queries: 1) the number of candidate points returned during the filter step, 2) the number of nodes accessed in the B^+-tree, and 3) the time taken (in milliseconds) to perform the query and return the final results. Often we express the ratio of candidates and nodes over the total number of points in the dataset and the total number of nodes in the B^+-tree, respectively, as this eliminates skewed results due to varying the dataset.

The first experiments are on synthetic datasets (uniform and clustered) so we can properly simulate specific dataset conditions, and we later apply these results towards evaluating strategies on real world dataset. All artificial datasets are given a specified number of points and dimensions in the unit space $[0.0, 1.0]$. For clustered data we provide the number of clusters and the standard deviation of the independent Gaussian distributions centered on each cluster (in each dimension). For each dataset, we randomly select 500 points as kNN queries (with $k = 10$) for all experiments, which ensures that our query point distribution follows the dataset distribution.

Sequential scan is often used as a benchmark comparison for worst-case performance. It must check every data point, and even though it does not use the B^+-tree for retrieval, total tree nodes provides the appropriate worst-case comparison. Note that all data fits in main memory, so all experiments are compared without depending on the behaviors of specific hardware-based disk caching routines. In real-life however, disk-based I/O bottlenecks are a common concern for

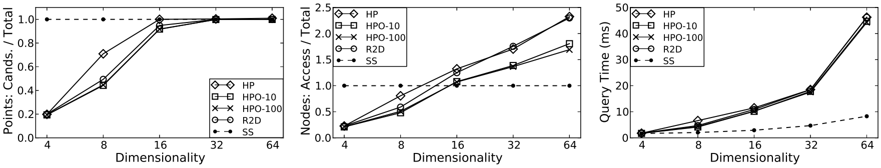


Fig. 2. Space-based methods on uniform data (10K) over dimensions

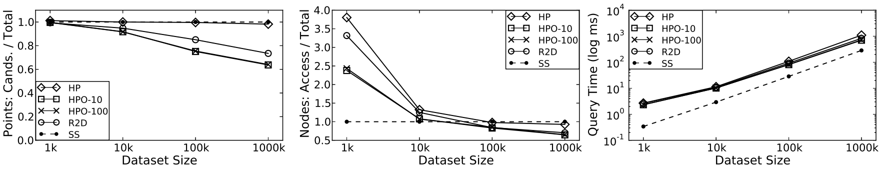


Fig. 3. Space-based methods on uniform data (16D) over dataset size

inefficient retrieval methods. Therefore, unless sequential scan runs significantly faster, there is a greater implied benefit when the index method does not have to access every data record, which could potentially be on disk.

4.1 Space-Based Strategies in Uniform Data

Our first experiments compare *Sequential Scan* (SS) to space-based methods in uniform datasets ranging from 4 to 64 dimensions and 1,000 (1k) to 1 million (1000k) points. We present *Half-Points* (HP) and *Half-Points-Outside* (HPO), specifically HPO-10 and HPO-100, and also show the RAND method with an equivalent $2d$ reference points (R2D).

Figures 2 and 3 validate the original claim that HPO performs better than HP [10,20], but surprisingly it also shows that R2D works better than HP. We can also see that a movement of 10.0 (HPO-10) outside of the dataspace is sufficient for performance improvements with HPO, and there is minimal gain thereafter. Although space-based methods take longer than SS in 16 dimensions (16D) or less, they access significantly less nodes and return fewer candidates. Note that it is possible to access the same nodes multiple times because data points from disjoint partitions can be stored in the same tree leaves. Another important performance factor is dataset size, shown in Figure 3 over a constant 16D. This can be linked to Figure 2 at 10k data points. We now log-transform the query time to show that as expected, larger datasets slow down all methods. However, sequential scan grows the fastest (with a linear increase), because at a certain point space-based strategies begin to properly filter the congested space and access less nodes while returning fewer candidates.

While still using uniform data, we investigate the effects of varying the number of reference points. Figures 4 and 5 look at the RAND method with 16 (R16), 64 (R64), 256 (R256), and 1024 (R1024) reference points. We also include dynamic methods of $2d$ over number of dimensions d (R2D) and $\sqrt{p}$ over number of points p (RP*), which are meant to better account for the specific dataspace characteristics. The results highlight the trade-off between dimensions

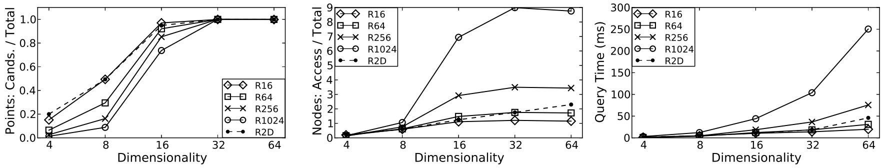


Fig. 4. Varying number of random ref. points on uniform data (10K) over dimensions

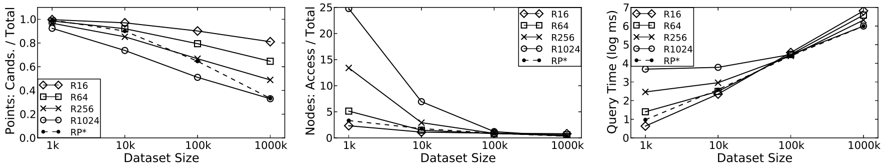


Fig. 5. Varying number of random ref. points on uniform data (10K) over dataset size

of a space and total points, showing that as the number of dimensions increase, more partitions reduce the number of candidates, but also increase the nodes accessed and overall query time. Conversely, as the number of data points increase and dimensionality holds constant, kNN queries become more compact, and the number of candidates and nodes decreases leading to a shorter query time.

4.2 The Transition to Clustered Data

Since most real-world data is not uniform, we turn our attention to clustered data and data-based partitioning strategies. As mentioned by the authors in the original iDistance publications [10,20], data-adaptive indexing is the primary strength of iDistance, and we too show it greatly improves overall performance. We start by trying to better understand when data-based strategies overtake space-based strategies through varying cluster densities in the space, which has not been investigated previously. For each dataset, cluster centers (12 total) are randomly generated and then points are sampled with a standard deviation (*stdev*) ranging from 0.40 to 0.005 in each dimension of the 16D space with a total of 100k points equally distributed among clusters. We use the actual cluster centers – *True Centers* (TC) – as the only reference points. For comparison, we include *Half-Points* (HP) and *Sequential Scan* (SS) as baseline benchmarks. The RAND method was not included because it produces unpredictable results depending on the location of reference points and underlying data clusters.

In Figure 6, we can see the effect that cluster density has as the space transitions from very loose to extremely tight clusters. We do not report candidates because the results closely mirror the nodes accessed ratio. While using the true centers of the clusters as reference points quickly becomes the better technique, it eventually stalls out and fails to improve once the data is sufficiently dense – but notice that HP's performance steadily increases to near similar results. Since the space-based reference points are not bound to clusters, they continue to increase in effectiveness by searching smaller and smaller "slices" of each partition.

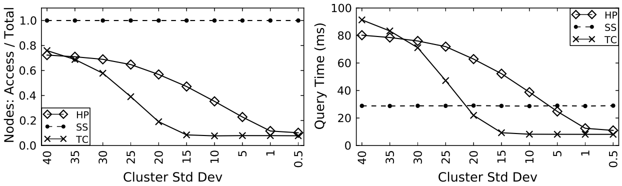


Fig. 6. Results over varying cluster density (by standard deviation)

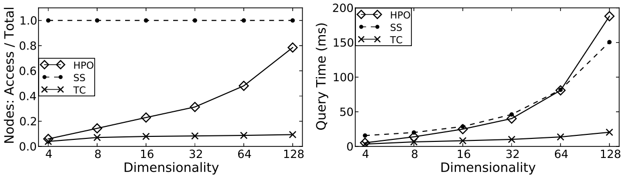


Fig. 7. Results over dimensions of 12 clusters with 0.1 standard deviation

We can further see these trade-offs in Figure 7. Here we set the *stdev* of all 12 clusters to 0.1 and vary the dimensionality of 100k data points. The 12 equal-sized clusters seem to explain why TC stabilizes with around 8% (or 1/12) of the nodes accessed in both of these figures. In other words, the clusters become so dense that although the *k*NN queries rarely have to search outside of a single partition, they ultimately have to search through the entire partition containing the query. We confirm this in Figure 8, which shows the total partitions checked and candidates returned for three clustered datasets with 6 (TC6), 12 (TC12), and 24 (TC24) clusters over varying cluster density. Notice that all three start with accessing all partitions and most data points, but all converge to only one checked partition with the respective ratio of candidates.

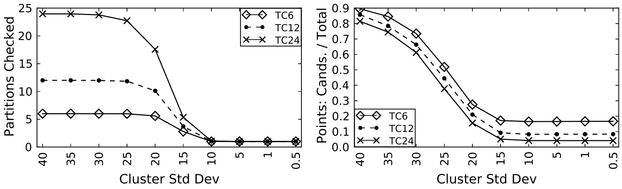


Fig. 8. Results of various numbers of clusters over cluster density

4.3 Reference Points: Moving from Clusters

We now investigate more advanced data-based partitioning strategies using the *True Centers* (TC) of clusters as our benchmark reference points. Original works make mention of reducing partition overlap, and thereby increasing performance, by moving reference points away from each other [10,20], but did not investigate it. This approach should perform better than TC because there will be less equi-distant points for each reference point, meaning the lossy transformation is less destructive for true data point similarity.

We present two strategies for moving reference points away from cluster centers. Since cluster centers are typically found by minimizing inter-cluster similarity while maximizing intra-cluster similarity, by moving reference points away from the cluster centers, one could hypothesize that there should be less equi-distant points in each partition and therefore a more discriminative one-dimensional B$^+$-tree index. The two methods are: 1) *Min-Edge* (M), moving towards the closest edge of the dataspace in any single dimension, and 2) *Random* (R), moving randomly in any/all dimensions. We specify movement by a total distance in the multi-dimensional dataspace and both methods are capable of pushing reference points outside of the dataspace – which makes the *Min-Edge* method similar to *Half-Points Outside* (HPO). Using *Min-Edge* on the data in Figure 1(b) as an example, the upper-left cluster center will decrease along the x-axis, and the upper-right cluster will increase along the y-axis.

Figure 9 shows the ratio of candidates returned from the two cluster center movement methods (M and R), with movement distances of {0.025, 0.05, 0.1, 0.2, 0.4}, each compared to TC. Each method performs best with a movement distance of 0.2, as shown with TC in the third column chart for better readability. We can see that above 16D (with 12 clusters and 100k points) no methods seem to make a significant difference. However, lower dimensions do support our hypothesis that moving away from the centers can help. Figure 10 shows the same methods in 16D over a varying number of data points, and here we see the methods also become ineffective as the number of points increase.

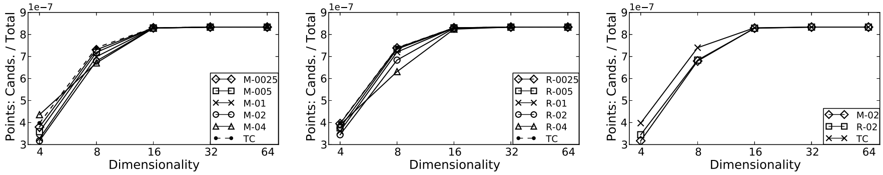


Fig. 9. Results of center movement methods

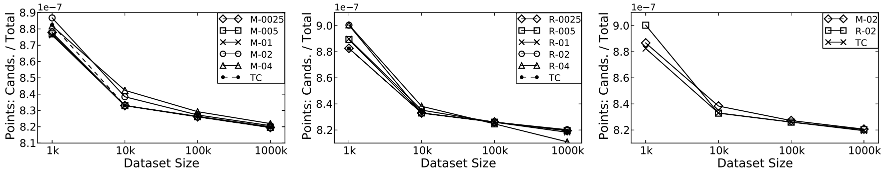


Fig. 10. Results of center movement methods

4.4 Reference Points: Quantity vs. Quality

While we know that iDistance performs well on datasets with known clusters, a more common scenario is less knowledge of the data where the size and number of clusters are unknown. This is the focus of the original iDistance works, which suggest the use of any popular clustering algorithm as a pre-processor to identify more optimal reference point placements. The original publications used BIRCH [23] in 2001 [20] and k-Means [12] in 2005 [10].

In these experiments we investigate the effect of the number of provided clusters during our pre-processing with the k-Means algorithm. It should be stated that k-Means is known to be sensitive to the initial starting position of cluster centers, and does not ensure any balance between cluster populations. We use a standard MATLAB implementation and mitigate these inherent weaknesses by initializing our cluster centers on a randomly sampled data subset, and forcing all clusters to contain at least one point so the resultant reference points are not accidentally removed and ignored from the space. Although never discussed in previous works, we believe it is very important to address the case of non-empty clusters, especially when analyzing how well a certain number of reference points perform. Otherwise, there is no guarantee that the specified number of reference points actually reflects the same number of partitions as intended.

The authors of iDistance originally suggested a general setting of $2d$ reference points – so k-Means with $k = 2d$ clusters – which also matches the space-based strategies [10,20]. In Figure 11, we look at the performance of k-Means (KM) with d-relative clusters from $d/2$ to $4d$, in various dimensions over 100k points in 12 clusters. We also include *True Centers* (TC) as our current baseline benchmark, and k-Means with 12 clusters but without knowledge of the true cluster centers upon initialization (KM-12*). Notice the relatively equal nodes accessed ratio for all methods in higher dimensions, but the increase in overhead time taken for the methods with more clusters (partitions).

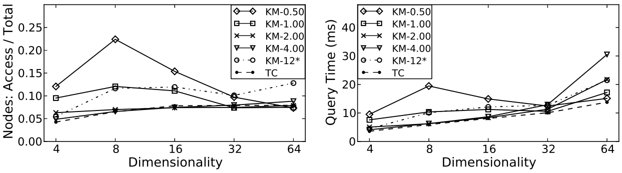


Fig. 11. Varying k-Means centers with 12 clusters (100K points) over dimensions

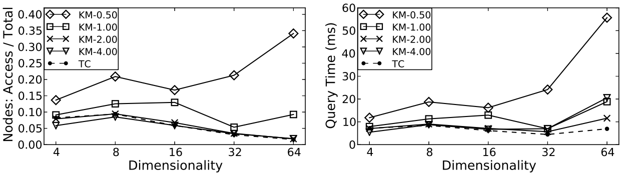


Fig. 12. Varying k-Means centers with d clusters (100K points) over dimensions

An important realization here is how beneficial the knowledge of true cluster centers can be, as we see TC performs more consistently (and nearly always better) than other methods over all dimensions. The same can be seen in Figure 12, where we now generate d clusters in the dataset instead of only 12. However, here we see that in higher dimensions more clusters make a major difference for the number of nodes accessed, and $2d$ clusters seem in many cases to be an appropriate balance between the number of partitions and the time taken to search all the partitions, as both $1d$ and $4d$ clusters are equally slower. Also note that

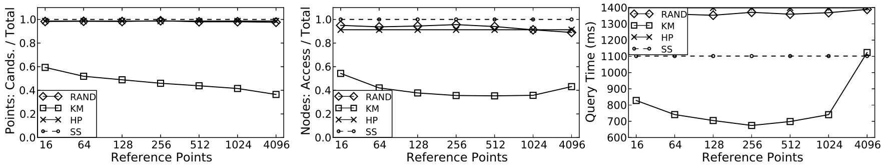


Fig. 13. Results of k-Means and RAND on real data over varying reference points

setting k to the number of known clusters for k-Means (KM-12* in Figure 11) does not guarantee performance because of the variability of discovered clusters from the k-Means algorithm.

Our final experiments use a real dataset to determine if any of our findings carry over from synthetic dataset studies. We use a popular real world dataset containing one million 128-dimensional SIFT feature vectors[2]. This dataset was recently used by the authors of the SIMP algorithm [17] to show comparatively better performance over their private implementation of iDistance using 5,000 reference points. However, without knowledge of algorithmic options, or several baseline experiments to show optimal performance results, we have very little insight into the effectiveness (and reproducibility) of their specific comparison.

In Figure 13, we look at RAND and k-Means over a varying number of reference points, and include HP and SS methods as comparisons. We can see that the only method that performs significantly better than SS is k-Means (KM). Although the number of candidates returned continues to decrease as we add reference points, we can see that after a certain point the overhead costs of additional partitions outweighs the filtering benefits, and the number of nodes accessed begins to increase while query time dramatically rises. We note there exists a clear range of relatively equivalent results from approximately $d/2$ (64) to $4d$ (512) partitions, which might be a combination of many factors including indexing performance and dataset characteristics. This performance plateau also provides an excellent measure for tuning to the proper number of partitions.

We also analyzed the number of partitions that were empty or checked for candidates, and the results of RAND exemplified our concerns over empty partitions and poor reference point placement. Essentially, as the number of random reference points increased, the more empty partitions are created. Worse yet, every non-empty partition is almost always checked due to high overlap and poor placement relative to each other and the data (and query) distributions.

5 Discussion

We see a promising result in Figure 10 at 1000k data points, suggesting that it is still possible to produce better results by moving reference points. This suggests there may exist a more sophisticated solution than the relatively simple methods we presented. We note that because of the default closest distance assignment strategy, when reference points are moved the points assigned to them may

[2] Publicly available at: `http://corpus-texmex.irisa.fr/`

change. Thus our efforts to reduce the number of equi-distant points may have been confounded, and if reference points are moved outside the dataspace, their partitions may become empty. Unfortunately, we found no significant difference in results by employing a static partition assignment before and after reference point movement, and therefore did not include the results for discussion. Clearly, more knowledge is required to move reference points in an optimal way that impacts partitioning efficiency. We have begun investigating the idea of *clustering for the sake of indexing*, by learning cluster arrangements explicitly for use as reference points within iDistance [19].

In general, we find a trade-off between dimensionality and dataset size, where more dimensions lead to less precise query regions (classic curse of dimensionality problem), but more points allow smaller regions to fully satisfy a given query. Space-based methods suffer much worse from dimensionality and are really not ideal for use. We agree with the original authors that $2d$ reference points seems appropriate as a general recommendation. In relatively moderate to small datasets and multi-dimensional spaces, $2d$ is probably overkill but far less burdensome than in exceptionally large datasets and high-dimensional spaces where the cost of additional reference points dramatically increases without providing much benefit. Results strongly support an intelligent data-centric approach to the amount and placement of reference points that results in minimally overlapping and non-empty partitions.

6 Conclusions and Future Work

We presented many complementary results to that of the original iDistance works, and through extensive experiments on various datasets and data characteristics we uncovered many additional findings that were not presented or discussed in prior works. This paper establishes a self-standing baseline for the wide variance in performance of partitioning strategies that opens the door for more directed and concise future works grounded on our findings. These results have also helped to establish an up-to-date benchmark and best practices for using the iDistance algorithm in a fair and efficient manner in application or comparative evaluations.

Many of the results show that traditional iDistance partitions stabalize in performance by accessing entire clusters (within single partitions), despite dataset size and dimensionality. This leads us to explore methods to further segment partitions in future work, so that we can better prune away large sections of dense data clusters. These novel strategies are much like the works of the iMinMax [13] and recently published SIMP [17] algorithms, whereby we can incorporate additional dataspace knowledge at the price of added complexity and performance overhead. Our preliminary work shows potential enhancements to the filtering power of iDistance through novel algorithm extensions that help reduce the negative effects of equi-distant points and partition overlap.

Acknowledgements. This work was supported in part by two NASA Grant Awards: 1) No. NNX09AB03G, and 2) No. NNX11AM13A. A special thanks to all research and manuscript reviewers.

References

1. Aurenhammer, F.: Voronoi diagrams – a survey of a fundamental geometric data structure. ACM Comput. Surv. 23, 345–405 (1991)
2. Bayer, R., McCreight, E.M.: Organization and maintenance of large ordered indices. Acta Informatica 1, 173–189 (1972)
3. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. SIGMOD Rec. 19, 322–331 (1990)
4. Bellman, R.: Dynamic Programming. Princeton University Press (1957)
5. Berchtold, S., Bhm, C., Kriegal, H.P.: The pyramid-technique: towards breaking the curse of dimensionality. ACM SIGMOD Record 27(2), 142–153 (1998)
6. Doulkeridis, C., Vlachou, A., Kotidis, Y., Vazirgiannis, M.: Peer-to-peer similarity search in metric spaces. In: VLDB 2007 (2007)
7. Guttman, A.: R-trees: a dynamic index structure for spatial searching. SIGMOD Rec. 14, 47–57 (1984)
8. Ilarri, S., Mena, E., Illarramendi, A.: Location-dependent queries in mobile contexts: Distributed processing using mobile agents. IEEE TMC 5, 1029–1043 (2006)
9. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proc. of the 30th Annual ACM Symposium on Theory of Computing, STOC 1998, pp. 604–613. ACM, New York (1998)
10. Jagadish, H.V., Ooi, B.C., Tan, K.L., Yu, C., Zhang, R.: iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. ACM Trans. Database Syst. 30(2), 364–397 (2005)
11. Lowe, D.: Object recognition from local scale-invariant features. In: The Proc. of the 7th IEEE Inter. Conf. on Computer Vision, vol. 2, pp. 1150–1157 (1999)
12. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: Cam, L.M.L., Neyman, J. (eds.) Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 281–297. University of California Press (1967)
13. Ooi, B.C., Tan, K.L., Yu, C., Bressan, S.: Indexing the edges: a simple and yet efficient approach to high-dimensional indexing. In: Proc. of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2000, pp. 166–174. ACM, New York (2000)
14. Qu, L., Chen, Y., Yang, X.: idistance based interactive visual surveillance retrieval algorithm. In: Intelligent Computation Technology and Automation (ICICTA), vol. 1, pp. 71–75 (October 2008)
15. Shen, H.T.: Towards effective indexing for very large video sequence database. In: SIGMOD Conference, pp. 730–741 (2005)
16. Shi, Q., Nickerson, B.: Decreasing Radius K-Nearest Neighbor Search Using Mapping-based Indexing Schemes. Tech. rep., University of New Brunswick (2006)
17. Singh, V., Singh, A.K.: Simp: accurate and efficient near neighbor search in high dimensional spaces. In: Proc. of the 15th Inter. Conf. on Extending Database Technology, EDBT 2012, pp. 492–503. ACM, New York (2012)

18. Tao, Y., Yi, K., Sheng, C., Kalnis, P.: Quality and efficiency in high dimensional nearest neighbor search. In: Proc. of the 2009 ACM SIGMOD Inter. Conf. on Mgmt. of Data, SIGMOD 2009, pp. 563–576. ACM, New York (2009)
19. Wylie, T., Schuh, M.A., Sheppard, J., Angryk, R.A.: Cluster analysis for optimal indexing. In: Proc. of the 26th FLAIRS Conf. (2013)
20. Yu, C., Ooi, B.C., Tan, K.L., Jagadish, H.V.: Indexing the Distance: An Efficient Method to KNN Processing. In: Proc. of the 27th Inter. Conf. on Very Large Data Bases, VLDB 2001, pp. 421–430. Morgan Kaufmann Publishers Inc., San Francisco (2001)
21. Zhang, J., Zhou, X., Wang, W., Shi, B., Pei, J.: Using high dimensional indexes to support relevance feedback based interactive images retrieval. In: Proc. of the 32nd Inter. Conf. on Very Large Data Bases, VLDB 2006, pp. 1211–1214 (2006)
22. Zhang, R., Ooi, B., Tan, K.L.: Making the pyramid technique robust to query types and workloads. In: Proc. 20th Inter. Conf. on Data Eng., pp. 313–324 (2004)
23. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: an efficient data clustering method for very large databases. SIGMOD Rec. 25(2), 103–114 (1996)

Extending High-Dimensional Indexing Techniques Pyramid and iMinMax(θ): Lessons Learned

Karthik Ganesan Pillai, Liessman Sturlaugson,
Juan M. Banda, and Rafal A. Angryk

Montana State University, Bozeman, Montana - 59717-3380, USA
{k.ganesanpillai,liessman.sturlaugson,
juan.banda,angryk}@cs.montana.edu

Abstract. Pyramid Technique and iMinMax(θ) are two popular high-dimensional indexing approaches that map points in a high-dimensional space to a single-dimensional index. In this work, we perform the first independent experimental evaluation of Pyramid Technique and iMinMax(θ), and discuss in detail promising extensions for testing k-Nearest Neighbor (kNN) and range queries. For datasets with skewed distributions, the parameters of these algorithms must be tuned to maintain balanced partitions. We show that, by using the medians of the distribution we can optimize these parameters. For the Pyramid Technique, different approximate median methods on data space partitioning are experimentally compared using kNN queries. For the iMinMax(θ), the default parameter setting and parameters tuned using the distribution median are experimentally compared using range queries. Also, as proposed in the iMinMax(θ) paper, we investigated the benefit of maintaining a parameter to account for the skewness of each dimension separately instead of a single parameter over all the dimensions.

Keywords: high-dimensional indexing, iMinMax, Pyramid Technique.

1 Introduction

Efficient indexing structures exist for storing and querying low-dimensional data. The B$^+$-tree [2] offers low-cost insert, delete, and search operations for single-dimensional data. The R tree [7] extends the concepts of the B$^+$-tree to 2 or more dimensions by inserting minimum bounding rectangles into the keys of the tree. The R* tree [3] improves the performance of the R tree by reducing the area, margin and overlap of the rectangles. Unfortunately, the performance of these hierarchial index structures deteriorates when employed to handle highly dimensional data [8]. On data with more than 8 dimensions, most of these techniques perform worse than a sequential scan of the data and this performance degradation has come to be called the “curse of dimensionality” [5]. Improved techniques for indexing highly dimensional data are necessary.

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 253–267, 2013.

One popular approach in addressing the problem of highly dimensional data is to employ an algorithm that maps the values of a high-dimensional record to a single-dimensional index [10]. After the data is collapsed to this single-dimensional index, it is possible to re-use existing algorithms and data structures that are optimized for handling single-dimensional data, such as the B^+-tree, which offers fast insert, update, and delete operations. An advantage of mapping to a single dimension and using a B^+-tree is that the algorithms can be easily implemented on top of an existing DBMS [4]. The most widely cited examples of this strategy include the Pyramid Technique [4], and the iMinMax(θ) [8].

Both the Pyramid Technique (PT) and iMinMax(θ) partition the data space into different partitions that map the high-dimensional data to a single dimensional value. To partition the data space, the PT uses the center point of the data space [4], and the iMinMax(θ) uses the dimension that has the largest or smallest value [8]. For data sets with skewed distributions, these two indexing techniques can result in a disproportionate number of points in the resulting partitions. In this paper, we show that by using medians (approximate and actual) of the skewed distribution in the data we can improve the partitioning strategy in a way that will better balance the number of points in resulting partitions.

The rest of the paper is organized as follows: In section 2, we give a brief background of PT and iMinMax(θ). In section 3 we explain PT and our proposed extensions in detail. In section 4 we explain iMinMax(θ) and our proposed extensions in detail. Finally, we present a large variety of experiments demonstrating the impact of extensions to PT and iMinMax(θ), and conclude with a summary of results.

2 Brief Background

The PT introduced in [4], partitions a data space of d dimensions into $2d$ hyperpyramids, with the top of each pyramid meeting at the center of the data space. The index of each point has an integer part and a decimal part. The integer part refers to the pyramid in which the point can be found, and the decimal part refers to the "height" of the point within that pyramid. Although two points may be relatively far apart in the data space, any number of points can potentially be mapped to the same index, because each point is indexed by a single real value.

Following a similar strategy, the iMinMax(θ), first presented in [8], maps each point to the "closest edge/limit" of the data space instead of explicitly partitioning the data space into pyramids. By mapping to axes instead of pyramids, they reduce the number of partitions from $2d$ to d. The simple mapping function was also intended to avoid more costly pyramid intersection calculations.

Mapping to a single dimension from multiple dimensions results in a lossy transformation. Therefore, both of these techniques must employ a filter and refine strategy. To be useful, the transformation should allow much of the data to be ignored for a given query. At the same time, the transformation must ensure that the filter step misses no true positives, so that, after the refine step removes the false positives, the result is *exactly* the points that match the query.

3 The Pyramid Technique

The PT [4] divides the high-dimensional data space to $2d$ pyramids (see Fig. 1), each of which share the center point of the data space as the top of the pyramid and have a $(d-1)$-dimensional surface of the data space as the base of the pyramid. A point in the data space is associated to a unique pyramid. The point also has a height within its pyramid. This association of point to a pyramid is called the pyramid number of a point in the data space. Any order-preserving one-dimensional index structure can be used to index pyramid values. For both insert and range query processing, the d-dimensional input is transformed into a 1-dimensional value, which can be processed by the B$^+$-tree. The leaf nodes of the B$^+$-tree store the d-dimensional point value and the 1-dimensional key value. Thus, inverse transformation is not necessary. In the next section, the data space partitioning will be explained in greater detail.

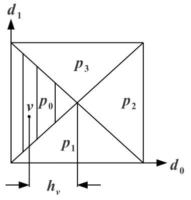


Fig. 1. Partition of data space into pyramids

3.1 Index Creation

The partition of the data space in the PT follows two steps. In the first step, the d-dimensional data space is divided into $2d$ pyramids, with the center point of the data space as the top of each pyramid and a $(d-1)$-dimensional surface as the base of each pyramid. In the second step, each pyramid is divided into multiple partitions, each partition corresponding to one data page of the B$^+$-tree. Fig. 1 shows the partitioning of a 2-dimensional data space into four pyramids p_0,p_1,p_2, and p_3, which all have the center point of the data space as the top and one edge of the data space as the base. Also the partitions within pyramid p_0, and height (h_v) of a point v in pyramid p_0 are shown in the figure.

The pyramid value of a point (pv_v), is the sum of the pyramid number and the height of the point within that pyramid. Calculation of the pyramid number and the height of a point is shown in *Algorithm* 1. In this algorithm, D is the total number of dimensions, d_{max} is the pyramid number in the data space partition. The algorithm assumes the data space has been normalized so that the center of the data space is at 0.5 in each dimension. Using the pv_v as a key, the d-dimensional point is inserted in the B$^+$-tree in the corresponding data page of the B$^+$-tree.

3.2 Query Processing

We now discuss point queries, range queries, and kNN queries in this section. The general definition of **point query**, can be stated as follows "Given a query

Algorithm 1. To calculate the pyramid value of a point v, adapted from [4]

```
PyramidValue(Point $v$)
$d_{max} = 0$
$height = |0.5 - v[0]|$
for $j = 1 \rightarrow D - 1$ do
   if $height < |0.5 - v[j]|$ then
      $d_{max} = j$
      $height = |0.5 - v[j]|$
   end if
end for
if $v[d_{max}] < 0.5$ then
   $i = d_{max}$
else
   $i = d_{max} + D$
end if
$pv_v = i + height$
return $pv_v$
```

point q, decide whether q is in the database." This problem can be solved by first finding the pyramid value pv_q of the query point q and querying the B$^+$-tree using pv_q. Thus, d-dimensional results are obtained sharing the pyramid value pv_q. From this result, we scan and determine whether the result contains q and output the result.

Second, for **range queries**, which are stated as "Given a d-dimensional interval $[q_{0_{min}}, q_{0_{max}}], \ldots, [q_{d-1_{min}}, q_{d-1_{max}}]$, determine the points in the database which are inside the range." Range query processing using the PT is a complex operation, a query rectangle of a range query might intersect several pyramids, and computation of the area of the interval is not trivial.

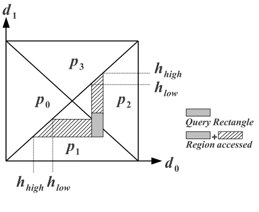


Fig. 2. Transformation of range queries

Fig. 2 shows a query rectangle and the region accessed for that query rectangle. This computation of the area is a two-step process. First, we need to determine which pyramids intersect with the query rectangle, and second, we need to determine the height intervals inside the pyramids. To determine the interval inside a pyramid (h_v between two values) for all objects is a one-dimensional indexing problem. Next, a simple point-in-rectangle test is performed in the refinement step.

An algorithm to find pyramid intersections and the interval within each pyramid for range queries is given in *Algorithm* 2, and it uses Equations 1 through

8. In this algorithm, the given query rectangle is first transformed in a way that the interval is defined relative to the center point. Next, pyramids in the data space partition that intersect with the query rectangle are found and the interval $[i + h_{low}, i + h_{high}]$ inside each intersected pyramid is computed. Using this interval, a point-in-rectangle test is performed using the B$^+$-tree.

$$intersect = \begin{cases} true & \text{if } k = D \\ false & \text{if } k \neq D \end{cases} \tag{1}$$

where k is obtained from equation 2

$$k = \begin{cases} \forall j, 0 \leq j < D, \hat{q}_{i_{min}} \leq -MIN(\hat{q}_j): & \text{if } i < D \\ \forall j, 0 \leq j < D, \hat{q}_{i-D_{min}} \geq -MIN(\hat{q}_j): & \text{if } D - 1 < i \end{cases} \tag{2}$$

Algorithm 2. To process range query $[\hat{qr}_{0_{min}}, \hat{qr}_{0_{max}}], ..., [\hat{qr}_{d-1_{min}}, \hat{qr}_{d-1_{max}}]$, adapted from [4]

```
RangeQuery($qr[D][2]$)
// Initialize variables
$sq[D][2]$
$qw[D][2]$
for $i = 0 \to D - 1$ do
    $sq[i][0] = qr[i][0] - 0.5$
    $sq[i][1] = qr[i][1] - 0.5$
end for
for $i = 0 \to (2D) - 1$ do
    if $(i < D)$ then
        $qw[i][0] = sq[i][0]$
        $qw[i][1] = sq[i][1]$
    end if
    if $(i \geq D)$ then
        $qw[i - D][0] = sq[i - D][0]$
        $qw[i - D][1] = sq[i - D][1]$
    end if
    // Using Equation 1
    if intersect then
        if $(i < D) \wedge (qw[i][1] > 0)$ then
            $qw[i][1] = 0$
        end if
        if $(i \geq D) \wedge (qw[i - D][0] < 0)$ then
            $qw[i - D][0] = 0$
        end if
        // Using Equation 6 and 8
        Find $h_{low}$ and $h_{high}$
        Search B$^+$-tree
    end if
end for
```

$$\bar{q}_j = \begin{cases} MIN(q_j) & \text{if MIN}(q_j) > \text{MIN}(q_i) \\ MIN(q_i) & otherwise \end{cases} \tag{3}$$

$$MIN(r) = \begin{cases} 0 & \text{if } r_{min} \leq 0 \leq r_{max} \\ min(|r_{min}|, |r_{max}|) & otherwise \end{cases} \tag{4}$$

$$MAX(r) = max(|r_{min}|, |r_{max}|) \tag{5}$$

$$h_{high} = MAX(\hat{q}_i) \tag{6}$$

$$hvalue = max_{0 \leq j < D} : (\bar{q}_j)(*) \tag{7}$$

$$h_{low} = \begin{cases} 0 & \text{if } \forall j, 0 \leq j < D : (\hat{q}_{j_{min}} \leq 0 \leq \hat{q}_{j_{max}}) \\ hvalue\ otherwise \end{cases} \tag{8}$$

Finally, we address ***k*NN** queries, which are stated as "Given a set S of n d-dimensional data points and a query point q, the kNN search is to find subset $S' \subseteq S$ of $k \leq n$ data points such that for any data point $u \in S'$ and $v \in S - S'$, $dist(u, q) \leq dist(v, q)$". The procedure to perform a kNN search using the decreasing-radius kNN search technique, introduced in [9], is given in *Algorithm* 3. In this method, after finding the pyramid number for the given query point q, the B$^+$-tree is searched to locate the leaf node that has the key value for the qiven point q, or the largest key value less than the key value of q. Once the key value is identified, the function $SEARCHLEFT$ ($SEARCHRIGHT$) is used to check the data points of the node towards the left (right) to determine if they are among the k nearest neighbors. When the difference between the current key value in the node and the pyramid value of q is greater than D_{max} and there are k data points in A or the key value of the leaf node is less (greater) than i ($i + 0.5$), the search on left (right) stops. In the next step, a query square W is generated to perform an orthogonal range search. The rest of the pyramids are examined one by one, and if a pyramid intersects with W, a $RangeQuery(W)$ is performed to check if the data points in this pyramid intersecting W are among the k nearest neighbors. The side length of W is updated after each pyramid is examined. The algorithm stops once all the pyramids have been examined.

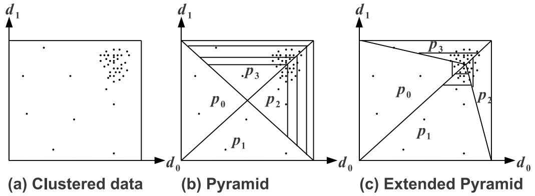
3.3 Extending Pyramid Technique

The data partitioning strategy of the original PT assumes a uniform data distribution. For clustered data, as shown in Fig. 3 (a), most of the data is contained in only a few pyramids. Partitioning this data space will result in sub-optimal space partition, as shown in Fig. 3 (b). A better partitioning approach is shown in Fig. 3 (c).

In the Extended PT, the basic idea is to let the pyramid's center point to follow the center of the actual data distribution. Thus, the data space is mapped to the canonical data space $[0, 1]^d$ such that the d-dimensional median of the data space is mapped to the center point. The transformation is applied only to determine the pyramid value of points and query rectangles, and hence an inverse transformation is again unnecessary.

Algorithm 3. The decreasing radius Pyramid kNN search algorithm, adapted from [9]

```
PyramidkNN($Point\ q, int\ k$)
$A \leftarrow emptyset$
$i \leftarrow pyramid\ number\ of\ the\ pyramid\ q\ is\ in$
$node \leftarrow LocateLeft(T, q)$
$SearchLeft(node, A, q, i)$
$SearchRight(node, A, q, i + 0.5)$
$D_{max} \leftarrow D(A_0, q)$
$Generate\ W\ centered\ at\ q\ with\ \triangle \leftarrow 2D_{max}$
for $j = 0 \rightarrow 2D - 1$ do
  if $(j \neq i) \wedge (W\ intersects\ pyramid\ j)$ then
    RangeQuery(W)
    $Update\ W\ with\ updated\ \triangle \leftarrow 2D_{max}$
  end if
end for
return $A$
```



(a) Clustered data (b) Pyramid (c) Extended Pyramid

Fig. 3. Extending Pyramid Technique for skewed data distributions

Computing the d-dimensional median is a hard problem [4] and thus two different approaches for finding the approximate median are explored. In the first approach, a histogram is created for each dimension. From the created histogram, the bin containing the median is found, and then that bin is searched for the median. This method requires n comparisons. In the second approach, we use the approximate median finding algorithm described in [1]. This method requires fewer than $(4/3) \times n$ comparisons and $(1/3) \times n$ exchanges on average, and fewer than $(3/2) \times n$ comparisons, with $(1/2) \times n$ exchanges in the worst case. The algorithm takes $n = 3^r$ as the input where r is an integer and proceeds in r stages. At each stage the algorithm divides the input into subsets of three elements, and calculates the median of each such triplet. Medians of each triplet are used in the next stage. The algorithm continues recursively, using the medians saved from the previous stage to compute the approximate median of the initial set. The method described can be generalized to array sizes which are not powers of three as follows: Let the array size be n, where $n = 3 \times t + k$, and where $k \in 0, 1, 2$. Then the first t triplets have their median extracted as before, and the t selected medians, as well as the remaining k elements are forwarded to the next stage. Moreover the direction in which those first triplets are selected alternates—either left-to-right or right-to-left. This ensures that no elements remain for more than one iteration. The algorithm continues iteratively using the results of each stage as input for the next one. This is done until the number of elements falls below a small fixed threshold. Finally, the elements are sorted to obtain their median as the final result.

The computation of the median can be done dynamically, in the case of dynamic insertions, or once in the case of a bulk load of points. This d-dimensional approximate median may lie outside the convex hull of the data cluster. Given the d-dimensional median mp_i of the data set, a set of d functions t_i are defined in [4], $0 \leq i < (d-1)$ transforming the given data space in dimension i such that the transformed data space still has a range of $[0,1]^d$, that the median of the data becomes the center point of the data space, and that each point in the original data space is mapped to a point inside the canonical data space.

$$t_i(x) = x^{-(1/log_2(mp_i))}$$

To insert a point v into an index, transform v into a point such that $v'_i = t_i(v_i)$ and determine the pyramid value $pv_{v'}$. Using $pv_{v'}$, point v is inserted into the B$^+$-tree. To process a query, first transform the query rectangle q into a query rectangle q' such that $q'_{i_{min}} = t_i(q_{i_{min}})$ and $q'_{i_{max}} = t_i(q_{i_{max}})$. Next, algorithms discussed in earlier sections are used to determine intersection of pyramids and ranges within pyramids to find the points in the query rectangle. Finally, the refine step is performed to filter out false positives as before.

$$q_j = \begin{cases} [j + \max_{i=1}^{d} x_{il}, j + x_{jh}] & \text{if } \min_{i=1}^{d} x_{il} + \theta \geq 1 - \max_{i=1}^{d} x_{il} \\ [j + x_{jl}, j + \min_{i=1}^{d} x_{ih}] & \text{if } \min_{i=1}^{d} x_{ih} + \theta < 1 - \max_{i=1}^{d} x_{ih} \\ [j + x_{jl}, j + x_{jh}] & \text{otherwise} \end{cases} \tag{9}$$

4 The iMinMax(θ) Algorithm

The iMinMax(θ) algorithm maps a d-dimensional space to a one-dimensional space, by indexing on the "edges". The maximum or minimum value among all the dimensions of a point is called an "edge" [8]. The iMinMax(θ) technique uses either the *Max edge* or *Min edge* in the index keys for the points.

4.1 Index Creation

As with the PT, the data is normalized such that each data point x resides in a unit d-dimensional space. A data point x is denoted as $x = (x_1, x_2, \ldots, x_d)$ where $x_i \in [0,1]\ \forall i$. Let $x_{max} = \max_{i=1}^{d} x_i$ and $x_{min} = \min_{i=1}^{d} x_i$. Each point is mapped to a single dimensional index value $f(x)$ as follows:

$$f(x) = \begin{cases} d_{min} + x_{min}, & \text{if } x_{min} + \theta < 1 - x_{max} \\ d_{max} + x_{max}, & \text{otherwise} \end{cases}$$

The parameter θ can be tuned to account for skewed data distributions in which much of the data would otherwise be mapped to the same edge, resulting in a less efficient search through the B$^+$-tree. In the simple case when $\theta = 0$, each point is mapped to the axis of the closest edge which is appropriate for uniformly distributed data. When $\theta > 0$, the mapping function is biased toward the axis of the maximum value, while $\theta < 0$ biases it toward the axis of the minimum value.

4.2 Query Processing

Range queries are first transformed into d one-dimensional subqueries. The range query interval for the jth dimension, denoted q_j, is calculated by Equation 9. The variables x_{il} and x_{ih} represent the low and high bound, respectively, for the range interval in the ith dimension. In the original iMinMax paper [8], they prove that the union of the results from the d subqueries is guaranteed to return the set of all points found within the range, while no smaller interval on the subqueries can guarantee this. Moreover, they prove that at most d subqueries must be performed. In fact, they prove that a subquery $q_i = [l_i, h_i]$ need not be evaluated if one of the following holds:

$$\min_{j=1}^{d} x_{jl} + \theta \geq 1 - \max_{j=1}^{d} x_{jl} \quad \text{and} \quad h_i < \max_{j=1}^{d} x_{jl}$$

$$\min_{j=1}^{d} x_{jh} + \theta < 1 - \max_{j=1}^{d} x_{jh} \quad \text{and} \quad l_i > \min_{j=1}^{d} x_{jh}$$

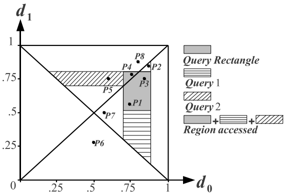


Fig. 4. Example iMinMax(θ) range query in 2 dimensions with $\theta = 0$

This occurs when all the answers for a given subquery are along either the *Max* or the *Min* edge. Thus, if either of these conditions hold, the answer set for q_i is guaranteed to be empty and can be ignored.

Fig. 4 shows the two subqueries generated by an example range query in 2 dimensions when $\theta = 0$. Query 1 returns $\{P1, P2, P3\}$ during the filter step and then refines by removing $P2$. Likewise, Query 2 returns $\{P4, P5\}$ and then refines by removing $P5$.

4.3 Tuning the θ Parameter

A d-dimensional median point can be used to calculate the optimal θ, denoted θ_{opt}. This d-dimensional median is calculated by first finding the median for each dimension. The combination of these d one-dimensional medians forms the median point used to calculate θ_{opt}. The x_{min} and x_{max} of this median are then used to calculate the optimal θ as

$$\theta_{opt} = 1 - x_{max} - x_{min}$$

4.4 Extending iMinMax(θ) with Multiple θ's

In this section, we investigate an extension to the iMinMax(θ) algorithm that incorporates multiple θ's, instead of using only the single θ parameter. For this extension, each dimension i will have its own unique parameter θ_i. The original θ parameter attempts to create a balanced partition of the data, making the median of the dataset a good choice from which to compute the θ parameter. This median, however, is calculated by finding the median of each dimensions individually. Instead of combining these medians into a single median and computing a single θ parameter, each of these medians is now used individually to compute each θ_i parameter.

This extension changes the mapping function when computing the single-dimension index value of a point. Notice that the mapping function compares only two dimensions at a time, the dimensions of the minimum and maximum values. The multiple θ_i parameters account for potentially different skewness across *pairs* of dimensions. Let θ_{min} and θ_{max} be the θ_i parameters for the dimensions of the minimum and maximum values, respectively. The mapping function now becomes:

$$f(x) = \begin{cases} d_{min} + x_{min}, & \text{if } x_{min} + \theta_{min} < 1 - x_{max} - \theta_{max} \\ d_{max} + x_{max}, & \text{otherwise} \end{cases}$$

The mapping of the range subqueries and the criteria for subquery pruning are modified similarly. The introduction of multiple θ_i parameters does not add significant overhead to index creation, with the number of θ_i parameters scaling linearly with the number of dimensions and with the mapping function only requiring one additional term.

5 Experiments

In this section we first present the experiments and results of PT and proposed extensions, followed by the experiments and results of iMinMax(θ) and proposed extensions, respectively. The benefits of using medians and the influence of different approximate median methods is discussed for PT and the effects of tuning the θ parameter and multiple θ's are discussed for iMinMax(θ).

5.1 Pyramid Technique (PT)

For the PT, we performed two different experiments on skewed distributions. For the first experiment, we measured the influence of approximate median methods on tree construction time and kNN query performance across different dimensions. For this experiment a total of three data sets are created with $4, 8$, and 16 dimensions and each data set has $500,000$ data points. For the second experiment, we measured the influence of approximate median methods on kNN query performance across different data set size. For this experiment a total of three data sets are created with $500,000$, $750,000$, and $1,000,000$ data points and each data set has 16 dimensions.

The results (see Figs. 5, 6, and 7) for the PT show how different approximate median methods influence tree construction time and kNN query performance. Each of the figures for PT shows four lines representing the various methods (see Table 1).

Table 1. Summary of experiments on Pyramid Indexing technique (Names of experiment are used also as legends in Figs. 5, 6, 7)

Name	Experiment Description
PT	Pyramid Technique
HMPT	Histogram-based approximate median PT
AMPT	Approximate median PT
TMPT	Brute force median PT

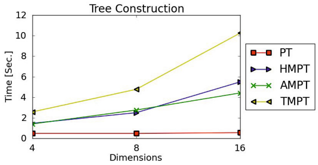


Fig. 5. Tree construction time over dimensionality of data space

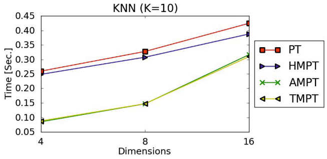


Fig. 6. kNN query retrieval behaviour over data space dimension

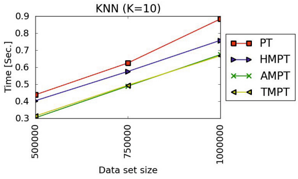


Fig. 7. kNN query retrieval time over database size

First, the benefit of using medians in PT and the influence of different approximate median methods is shown in Fig. 6 and 7 for kNN query on single clustered data sets with different dimensions and on single clustered data sets with different sizes. Each datapoint on the two graphs represents query retrieval time averaged over $1,000$ kNN ($k = 10$) random queries. The centroid of the cluster is offset from the center of the space, while the original PT does not account for this skewed distribution in the data. On the other hand, the other methods compute the d-dimensional (approximate) median to be the center of the data space.

This results in better space partitioning for the single-cluster datasets, because data points in the data space are more evenly mapped to different pyramids (see Fig. 3 (c)), thus improving the performance of the kNN queries. Moreover, from the figures we can observe that AMPT performance is close to TMPT in comparison to HMPT. AMPT, with its high probability of finding an approximate median within a very small neighborhood of the true median [1], leads to better space partitioning, hence results in better performance. These results demonstrate the benefit of using approximate median methods with PT on a single skewed distribution.

However, computing the exact or approximate median increases the time to build the index and we can observe this from Fig. 5. In order to reduce the time to build index for extended PT, we should use an approximate median method that is computationally less expensive.

5.2 iMinMax(θ)

For the iMinMax(θ), we performed two different experiments. For the first experiment, we measured the influence of setting a θ value on range query performance across different dimensions. For this experiment a total of three skewed distribution data sets are created with $4, 8$, and 16 dimensions and each data set has $500,000$ data points with single cluster. For the second experiment, we measured the influence of calculating a unique θ_i for each dimension i on range query performance across different dimensions. For this experiment a total of three data sets are created with $4, 8$, and 16 dimensions and each data set has $500,000$ data points. Moreover, for this experiment all the data set have ten clusters and each cluster has $50,000$ data points. To measure the performance of range queries, two different range queries are generated. For the narrow range query we picked $1,000$ random query points from the data set and modified a single dimension value of each point with +/- 0.01, and for the wide range query we selected $1,000$ random query points from the data set and modified half of the dimensions value of each point with +/- 0.01.

The results (see Figs. 8 through 13) for the iMinMax(θ) show how different values for the θ parameter (see Table 2) influence range query performance. The results show that, by setting θ as calculated using either the exact or approximate median, the performance improves upon the default $\theta = 0$ as the number of dimensions increases.

Effects of Tuning the θ Parameter. The benefit of tuning θ is shown in Fig. 8 and 9 for the narrow range query and wide range query with the single-cluster dataset. Each datapoint on the two graphs represents the query time averaged over $1,000$ range queries each centered at random points drawn from

Table 2. Summary of experiments on iMinMax(θ) Indexing technique (Names of experiment are used also as legends in Figs. 8 through 13)

Name	Experiment Description
DEF	Default setting when $\theta = 0$, which assumes that the center of the space is the centroid of the data
OPT	Single θ calculated by finding the exact median of the dataset
APP	Single θ calculated by finding the approximate median
MOPT	Multiple θ's calculated by finding the exact median for each dimension separately
MAPP	Multiple θ's calculated by finding the approximate median for each dimension separately

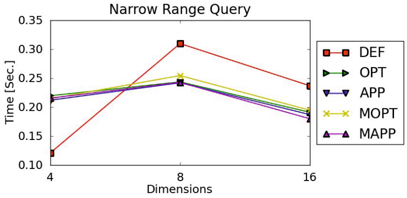


Fig. 8. Narrow range query retrieval time on tuning θ for single cluster

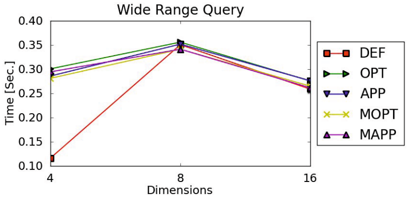


Fig. 9. Wide range query retrieval time on tuning θ for single cluster

the dataset. The centroid of the cluster is offset from the center of the space, while the default $\theta = 0$ does not account for this skewed distribution in the data. On the other hand, the other methods that calculate either one or multiple θ's are able to account for the skewed distribution and and are able to better optimize the range queries as the number of dimensions increases. With this dataset, the medians for the different dimensions do not vary significantly, and thus the difference between the single and multiple θ's is minimal.

Effects of Multiple θ_i's. The benefit of calculating a unique θ_i for each dimension i is shown in Figs. 10 and 11 for the narrow range query and for the wide range query with the ten-cluster dataset. Again, each datapoint on the two graphs represents the query time averaged over $1,000$ range queries each centered at random points drawn from the dataset. For this dataset, the medians of different dimensions vary by a difference of up to 41% of the space. This time the differences between the single θ and multiple θ's become more apparent. By maintaining a θ_i for each dimension, the $MOPT$ and $MAPP$ methods perform as well or better than the OPT and APP methods using the single θ.

On the other hand, calculating the exact or approximate median does not add significant time complexity to the creation and population of the B$^+$-tree, as shown by Figs. 12 and 13 for the single-cluster and ten-cluster datasets. These results demonstrate the benefit of calculating θ from the median of the dataset

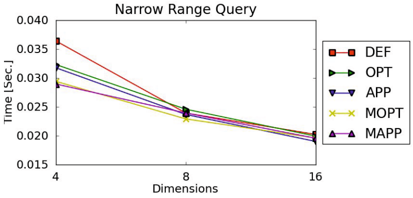


Fig. 10. Narrow range query retrieval time on calculating a unique θ_i for ten cluster

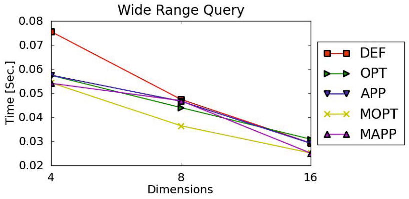


Fig. 11. Wide range query retrieval time on calculating a unique θ_i for ten cluster

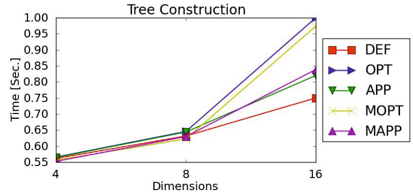


Fig. 12. Tree construction time over dimensionality of data space for single cluster

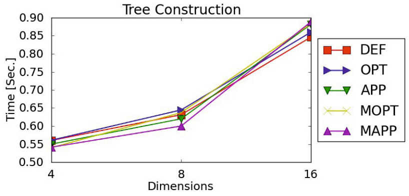


Fig. 13. Tree construction time over dimensionality of data space for ten cluster

with a skewed distribution and the benefit of calculating θ_i from the median of each dimension i of the dataset when the magnitude of the skew varies across different dimensions.

6 Conclusions

From the experiments with the PT (Figs. 6 and 7), it is demonstrated that using the d-dimensional median to be the center of the data space and mapping the given skewed data set to the canonical data space $[0, 1]^d$ results in better space partitioning, thus improving the performance of query retrieval time. Furthermore the influence of computational complexity (Fig. 5) and closeness to median of approximate median methods is demonstrated from the experiments.

From the experiments with the iMinMax(θ) algorithm (Figs. 8 and 9), it is demonstrated that deriving the θ parameter from the median offers improved range query performance over the default parameter setting for skewed datasets. Furthermore, calculating a unique θ_i for each dimension i (Figs. 10 and 11) can improve the performance of range queries for datasets with skewness that varies across different dimensions with little extra computational effort.

In this paper, we have shown experimentally for both PT and iMinMax(θ) algorithm, that by using the median (approximate) of the skewed distributions in the data, we can partition the data space into different partitions with proportionate number of points in each partition. Effectiveness is expected to increase as the dimensionality and data volume increases for skewed data distributions.

In future works, we plan to evaluate the performance of PT and iMinMax(θ) with their extensions on larger data sets, and more dimensions. We also, plan to compare our proposed extensions of PT and iMinMax(θ) algorithms with the approach described by Günnemann *et al.* in [6], as they also address indexing high-dimensional data that have skewed distributions.

Acknowledgments. This work was supported by two National Aeronautics and Space Administration (NASA) grant awards, 1) No. NNX09AB03G and 2) No. NNX11AM13A.

References

1. Battiato, S., Cantone, D., Catalano, D., Cincotti, G., Hofri, M.: An efficient algorithm for the approximate median selection problem. In: Bongiovanni, G., Petreschi, R., Gambosi, G. (eds.) CIAC 2000. LNCS, vol. 1767, pp. 226–238. Springer, Heidelberg (2000)
2. Bayer, R., McCreight, E.M.: Organization and maintenance of large ordered indexes. Acta Informatica 1, 173–189 (1972), doi:10.1007/BF00288683
3. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. SIGMOD Rec. 19, 322–331 (1990)
4. Berchtold, S., Böhm, C., Kriegel, H.-P.: The pyramid-technique: Towards breaking the curse of dimensionality. SIGMOD Rec. 27, 142–153 (1998)
5. Berchtold, S., Keim, D.A.: High-dimensional index structures database support for next decade's applications. In: Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, SIGMOD 1998, p. 501. ACM, New York (1998)
6. Günnemann, S., Kremer, H., Lenhard, D., Seidl, T.: Subspace clustering for indexing high dimensional data: a main memory index based on local reductions and individual multi-representations. In: Proceedings of the 14th International Conference on Extending Database Technology, EDBT/ICDT 2011, pp. 237–248. ACM, New York (2011)
7. Guttman, A.: R-trees: a dynamic index structure for spatial searching. SIGMOD Rec. 14, 47–57 (1984)
8. Ooi, B.C., Tan, K.-L., Yu, C., Bressan, S.: Indexing the edges – a simple and yet efficient approach to high-dimensional indexing. In: Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2000, pp. 166–174. ACM, New York (2000)
9. Shi, Q., Nickerson, B.: Decreasing Radius K-Nearest Neighbor Search Using Mapping-based Indexing Schemes. Technical report, University of New Brunswick (2006)
10. Yu, C., Bressan, S., Ooi, B.C., Tan, K.-L.: Querying high-dimensional data in single-dimensional space. The VLDB Journal 13, 105–119 (2004)

A Game Theory Based Approach for Community Detection in Social Networks

Lihua Zhou[1], Kevin Lü[2], Chao Cheng[1], and Hongmei Chen[1]

[1] Department of Computer Science and Engineering, Yunnan University,
Kunming 650091, China
[2] Brunel University, Uxbridge, UB8 3PH, UK
{lhzhou,hmchen}@ynu.edu.cn, 136388340@qq.com,
jiang.jianglu@gmail.com

Abstract. The attribute information of individuals, such as occupation, skill, faith, hobbies and interests, etc, and the structure information amongst individuals, such as mutual relationships between individuals, are two key aspects of information that are used to study individuals and communities in social networks. Considering only the attribute information or the structure relationship alone is insufficient for determining meaningful communities. In this paper, we report an on-going study, we propose an approach that incorporates the structure information of a network and the attribute information of individuals by cooperative games, and game theory is introduced to support strategic decision making in deciding how to recognize communities in social networks, such networks are featured by large number of members, dynamic and with varied ways of connections. This approach provides a model to rationally and logically detect communities in social networks. The Shapley Value in cooperative games is adopted to measure the preference and the contribution of individuals to a specific topic and to the connection closeness of a coalition. We then proposed an iterative formula for computing the Shapley Value to improve the computation efficiency, related theoretical analysis has also been performed. Finally, we further developed an algorithm to detect meaningful communities.

Keywords: social networks, community detection, game theory, Shapley Value.

1 Introduction

Community detection is an important issue in social network analysis, because communities, i.e. groups of individuals that are densely interconnected but only sparely connected with the rest of the network (Boccaletti et al. 2006; Guimera and Amaral 2005), are supposed to play special roles in the structure-function relationship (Shi et al. 2012), and thus the detected communities can be used in collaborative recommendation, information spreading, knowledge sharing, and other applications, which can benefit us greatly (Zhao et al. 2012).

Based on the prosperity that a community should have more internal than external connections, community detection aims at dividing the individuals of a network into

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 268–281, 2013.

some groups, such that the connections inside these groups are dense and the connections that run between individuals in different groups are sparse, therefore most existing studies on community detection mainly focus on the structure information of networks and overlook the attribute information of individuals. Communities detected by these works can reflect the strength of social relation, but they often lack unambiguous meanings, for example, members within a community may have different topics of interest (Zhao et al. 2012). On the contrary, classical clustering in data mining and machine learning paradigms aims at grouping individuals into classes or clusters by analyzing the attribute information of individuals, so that the minimal similarity between individuals within a cluster is higher than the maximal similarity between individuals in different clusters (Han and Kamber 2000). Clusters identified by classical clustering often have unambiguous meanings, for example, each cluster has one common topic, but these clusters can not reflect the strength of social relations. In fact, in the context of social networks, both the structure information and the attribute information are important for community detection (Zhao et al. 2012). Taking the attribute information or the structure information alone is insufficient in determining meaningful communities, because these manners may result in that the detected communities either have sparse connections between members within the same group, or lack unambiguous meanings.

In this paper, we propose a game theory based approach for detecting meaningful communities in social networks. This approach incorporates the structure information of networks and the attribute information of individuals. Game theory (Nash 1951) is a mathematical framework that describes interactions between multiple players (i.e. individuals) and allows for reasoning about their outcomes. In the context of social networks, the co-operations co-exist with the conflicts amongst individuals, because individuals with similar interests are more likely to cooperate with each other, but an individual's influence in a network is dependent on itself as well as others. This interactive and cooperative information can be analyzed by applying game theory.

Game theory is divided into two branches, called the non-cooperative (Nash 1951) and the cooperative (Zlotkin and Rosenschein 1994) branches. The non-cooperative game specifies various actions that are available to the players while the cooperative game (Zlotkin and Rosenschein 1994) describes the payoff that result when the players come together in different combinations. In a cooperative game, each combination of players is called a coalition, and a player's contribution to a coalition can be measured by the Shapley Value (Shapley 1953). It is a classical normative solution concept that provides a unique and fair solution to the cooperative game. In the context of social networks, meaningful communities are groups of individuals that are densely interconnected and have unanimity topics. Therefore, communities can be described by coalitions, and each individual's contribution to the closeness of connection and its preference to a specific topic can be evaluated by the Shapley Value.

It is known that the computation of the Shapley Value may be very hard even if the number of players is quite small. However, the number of individuals is large and the number of combinational coalitions is very large, the computation of Shapley Values for each player with respect to various coalitions is a problem. To improve the computation efficiency, we proposed an iterative formula for computing the Shapley

Value. Related theoretical analysis has also been performed. Based on the iterative formula, the Shapley Values for each player can be easily computed when we merge small-size coalitions into large-size coalitions.

To find the desired communities, i.e. coalitions in which the Shapley Value of every player is larger than those of the players in other coalitions, we propose an algorithm to combine small-size coalitions to large-size coalitions step by step. This manner of recursive combination can naturally reveal overlapped (Li et al. 2008) and hierarchical (Ravasz et al. 2002) structures of networks. The former means that vertices simultaneously belong to several groups, and the later means that communities are recursively grouped into a hierarchical structure.

The rest of the paper is organized as follows. Related work is introduced in section 2 and the game theory based approach for community detection is proposed in section 3. Section 4 concludes the paper.

2 Related Work

In data mining and machine learning paradigms, classical clustering has been researched for many years, so there are many clustering methods have been proposed from various perspectives, such as *k*-means, BIRCH, DBSCAN, STING and EM (Han et al. 2000). In the context of social networks, McCallum et al. (2005) presented the Author-Recipient-Topic model to discover the discussion topics; Tian et al. (Tian et al. 2008) proposed OLAP-style aggregation strategies to partition the graph according to attribute similarity. These methods were established mainly on the attribute information owned by the data objects themselves, instead of the relationship amongst them.

Many community detection algorithms, such as the Kernighan-Lin algorithm (Kernighan and Lin, 1970), fast algorithm (Newman 2004), nonnegative matrix factorization (NMF) algorithms (Wang et al. 2011), the metagraph factorization algorithms (Lin et al. 2011), and multi-objective evolutionary algorithm (Shi et al. 2012), detect communities by analyzing topological structure of networks. These algorithms did not take overlap or hierarchy of communities into account. Palla et al. (2005) proposed clique percolation method (CPM) to find overlapping communities, and Clauset et al. (2008) described the hierarchical organization of a graph by introducing a class of hierarchical random graphs. Ahn et al. (2010) reveal hierarchical and overlapping relationships simultaneously by reinventing communities as groups of links rather than nodes. However, these methods do not involve the attribute information.

The works that combined attribute information and structure information include the group detection algorithm (GDA) (Kubica et al. 2002), the group-topic (GT) model (Wang et al. 2005), and the topic oriented community detection approach (Zhao et al. 2012). In addition to the relations between individuals, the GDA also considered attributes of an individual, while the GT model considered the attributes of the relations (for example, the text associated with the relations). The topic oriented community detection approach clustered topics of individuals before analyzing topological structures of networks. Although the attribute information we used in this paper is the preference of each individual to topics, but unlike the topic oriented community detection approach, we incorporate attribute information and structure information rather than deal with them separately.

As an important branch of game theory, cooperative game theory has been used in various applications, such as the multi-objective categorization (Liu et al. 2011), the analysis of communications in wireless networks (Saad et al. 2009), the cooperation in mobile social networks (Niyato et al. 2011), and the measurement of the importance of individual nodes in networks (Gomez et al. 2003; Suri and Narahari 2008; Moretti et al. 2010), etc. In these researches, different problems are solved based on cooperative game theory.

To improve the computation efficiency of the Shapley Value, Owen (1972) proposed the multi-linear extension method, Fatima et al. (2008) proposed the linear approximation method, Castro et al. (2009) develop a polynomial method based on sampling theory to estimate the Shapley Value, Ieong and Shoham (2005) proposed concise representations for coalitional games, and Karthik et al. (2010) presented the study of the Shapley Value for network centrality. Based on these methods, one can gain efficient Shapley Value computation.

3 A Game Theory Based Approach for Meaningful Community Detection

In this section, after briefly giving the definitions of cooperative games and the Shapley Value, we introduce a game theory based approach for meaningful community detection, and we then develop the iterative formula for computing Shapley Value and present an algorithm to detect communities.

3.1 Cooperative Games and the Shapley Value

A cooperative game is a game where groups of players may enforce cooperative behaviour; therefore the game is a competition between coalitions of players, rather than between individual players. By cooperation, players gain more payoff than those players on their own. A cooperative game consists of two elements: (i) a set of players, and (ii) a characteristic function. Their meaning is presented in the definition 1.

Definition 1. *Cooperative games* (Liu et al. 2007). A *cooperative game* is a pair (N, v), where $N = \{1,2,...,n\}$ denotes a finite set of players and $v : 2^N \to R$ is the *characteristic function*, assigning a real value to each $S \subseteq N$. S (i.e. a group of players) is called a *coalition* and $v(S)$ is called the *payoff* of this coalition. $v(S)$ represents the value created when the members of S come together and interact. In general, v satisfies:

(1) $v(\Phi) = 0$.

(2) Superadditivity, i.e. $\forall S_1, S_2 \subseteq N$, if $S_1 \cap S_2 = \Phi$, then $v(S_1) + v(S_2) \le v(S_1 + S_2)$. This means that the value of a union of disjoint coalitions is no less than the sum of the coalitions' separate values.

(3) Monotonicity, i.e. if $S_1, S_2 \subseteq N$, $S_1 \subseteq S_2$, then $v(S_2) \geq v(S_1)$. This means that larger coalitions gain more.

In cooperative games, an important issue is how to allocate the payoff among the players in some fair way. Players will leave a coalition if they receive fewer payoffs than they expect to gain from the coalition. The *Shapley Value* (Shapley 1953) is a widely used value division scheme in the theory of coalitional games, because it provides a unique and fair solution.

Definition 2. The Shapley Value (Shapley 1953). The Shapley Value of player i in coalition $S \subseteq N$ with respect to (N, v) is given by formula (1):

$$\forall i \in S,\ \ SH_v(S,i) = \sum_{\{W \subseteq S | i \in W\}} \frac{(|W|-1)!(|S|-|W|)!}{|S|!}[v(W) - v(W - \{i\})] \tag{1}$$

Where $|S|$ (resp. $|W|$) is the cardinality of the set $|S|$ (resp. $|W|$), $v(W) - v(W - \{i\})$ defines the marginal contribution of player i to the coalition $|W|$, $\frac{(|W|-1)!(|S|-|W|)!}{|S|!}$ denotes the probability distribution function of all subsets of S. Therefore, $SH_v(S,i)$ reflects how much that player i contributes to the coalition S and denotes the average of player i's marginal contribution to all possible coalitions. A player who never adds much has a small Shapley Value, while a player that always makes a significant contribution has a high Shapley Value.

$SH_v(S,i)$ satisfies the following three axioms:

(1) The order of the players in S does not influence $SH_v(S,i)$.

(2) $\sum_i SH_v(S,i) = v(S)$·

(3) For any (N, v) and (N, v'), $SH_v(S,i) + SH_{v'}(S,i) = SH_{(v+v')}(S,i)$.

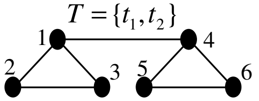
3.2 A Cooperative Game Model for Community Detection

Definition 3. A cooperative game model for community detection. Let $N = \{1, 2, ..., n\}$ be the set of all individuals in a social network, $A = (a_{ij})_{n \times n}, i, j \in N$ be the adjacency matrix, where $a_{ij} = 1, i, j \in N$ if a relationship exists between vertex i and j, otherwise $a_{ij} = 0, i, j \in N$, $T = \{t_1, t_2, ..., t_m\}$ be the set of finite topics, $P = (p_{it})_{n \times m}, i \in N, t \in T$ be the preference matrix, where p_{it} be the preference of player i with respect to the *t-th* topic, then the cooperative game model for community detection is defined as $CCG = (N, v)$, where players participating the cooperative game are individuals in N, a *coalition* S is a group of players, i.e. $S \subseteq N$, and the characteristic function $v_t(S)$ with respecting to the topic t ($t \in T$) is defined by following formula (2):

$$v_t(S)=\begin{cases} v_t(\Phi)=0 \\ v_t(S)=\sum_{i\in S}(\sum_{j\in S}\frac{a_{ij}}{d(i)}+p_{it}), \quad S\subseteq N, \quad t\in T \end{cases} \tag{2}$$

Where $d(i)=\sum_{j\in N}a_{ij}$, it is the degree of individual i . If $d(i)=0$ or $i=j$, then $\frac{a_{ij}}{d(i)}=0$. $v_t(S)\geq 0$ because $\frac{a_{ij}}{d(i)}\geq 0$ and $p_{it}\geq 0$. $v_t(S)$ incorporates the structure information and the attribute information by assigning a 'payoff' to each subset (coalition) of individuals in N . The payoff of a coalition represents the overall magnitude of the correlation between members of the coalition and of the members' preference to the *t-th* topic.

Example 1. A simple social network, shown in Fig.1, consists of 6 individuals (vertices : they represent 2 teachers (vertex 1,4) and 4 students (vertex 2,3,5,6) in a department), 7 relationships (edges: colleague relations between vertex 1 and 4; teachers and students relations between vertex 1 and 2, 1 and 3, 4 and 5, 4 and 6; schoolmate relations between vertex 2 and 3, 5 and 6), and 2 topics(t_1 : data mining; t_2 : artificial intelligence). Matrix *P* measures research interests of individuals on data mining and artificial intelligence. The measurement values can be obtained by analyzing attribute information of individuals, such as major, publications, attendant conferences, visited web sites, etc. Parts of the values of characteristic functions with respect to 2 topics are given in Tab.1.



$$P^T=\begin{array}{c|c} \begin{bmatrix} 0.5 & 1 & 1 & 0.5 & 0 & 0 \\ 0.5 & 0 & 0 & 0.5 & 1 & 1 \end{bmatrix} & \begin{matrix} t_1 \\ t_2 \end{matrix} \end{array}$$

Fig. 1. A simple social network

Table 1. Parts of the values of characteristic functions with respect to topic t_1 and t_2

$v_{t_1}(\{1,2\})$	$v_{t_1}(\{1,4\})$	$v_{t_1}(\{1,2,3\})$	$v_{t_1}(\{1,2,4\})$	$v_{t_1}(\{2,3,4\})$	$v_{t_1}(\{1,2,3,4\})$
2.33	1.67	5.17	3.5	3.5	6.33
……	……	……	……	……	……
$v_{t_2}(\{1,2\})$	$v_{t_2}(\{1,4\})$	$v_{t_2}(\{1,2,3\})$	$v_{t_2}(\{1,2,4\})$	$v_{t_2}(\{2,3,4\})$	$v_{t_2}(\{1,2,3,4\})$
1.33	1.67	3.17	2.27	1.5	4.33
……	……	……	……	……	……

$v_t(S)$ satisfies the Theorem 1~3.

Theorem 1. $v_t(S)$ satisfies Superadditive, i.e. $\forall S_1, S_2 \subseteq N$, if $S_1 \cap S_2 = \Phi$, then $v_t(S_1) + v_t(S_2) \le v_t(S_1 + S_2)$.

Proof: Let $i_1 ..., i_m, j_1, ..., j_n \in N$, $t \in T = \{t_1, t_2, ..., t_m\}$, $S_1 = \{i_1, ..., i_m\}$, $S_2 = \{j_1, ..., j_n\}$, then

$$v_t(S_1 + S_2) = \sum_{i \in S_1} \Big(\sum_{j \in S_1 + S_2} \frac{a_{ij}}{d(i)} + p_{it} \Big) + \sum_{j \in S_2} \Big(\sum_{i \in S_1 + S_2} \frac{a_{ji}}{d(j)} + p_{jt} \Big)$$

$$= \sum_{i \in S_1} \Big(\sum_{j \in S_1} \frac{a_{ij}}{d(i)} + p_{it} \Big) + \sum_{i \in S_1} \sum_{j \in S_2} \frac{a_{ij}}{d(i)} + \sum_{j \in S_2} \Big(\sum_{i \in S_2} \frac{a_{ji}}{d(j)} + p_{jt} \Big) + \sum_{j \in S_2} \sum_{i \in S_1} \frac{a_{ji}}{d(j)} .□$$

$$= v_t(S_1) + v_t(S_2) + \sum_{i \in S_1} \sum_{j \in S_2} \frac{a_{ij}}{d(i)} + \sum_{j \in S_2} \sum_{i \in S_1} \frac{a_{ji}}{d(j)} \ge v_t(S_1) + v_t(S_2)$$

Theorem 2. $v_t(S)$ satisfies monotonicity, i.e. if $S_1, S_2 \subseteq N$, $S_1 \subseteq S_2$, then $v_t(S_2) \ge v_t(S_1)$.

Proof: Let $i_1 ..., i_m \in N$, $S_1 = \{i_1, ..., i_n\}, S_2 = \{i_1, ..., i_m\}, n \le m$, then

$$v_t(S_2) = \sum_{i \in S_2} \Big(\sum_{j \in S_2} \frac{a_{ij}}{d(i)} + p_{it} \Big)$$

$$= \sum_{i \in S_1} \Big(\sum_{j \in S_1} \frac{a_{ij}}{d(i)} + p_{it} \Big) + \sum_{i \in S_1} \sum_{j \in S_2 - S_1} \frac{a_{ij}}{d(i)} + \sum_{j \in S_2 - S_1} \sum_{i \in S_1} \frac{a_{ji}}{d(j)} + \sum_{j \in S_2 - S_1} \Big(\sum_{i \in S_2 - S_1} \frac{a_{ji}}{d(j)} + p_{jt} \Big) □$$

$$= v_t(S_1) + \sum_{i \in S_1} \sum_{j \in S_2 - S_1} \frac{a_{ij}}{d(i)} + \sum_{j \in S_2 - S_1} \sum_{i \in S_1} \frac{a_{ji}}{d(j)} + v_t(S_2 - S_1) \ge v_t(S_1)$$

Theorem 3. If $S \subseteq N$, $i \in S$, the $v_t(S) - v_t(S - \{i\}) = \sum_{j \in S - \{i\}} \Big(\frac{a_{ij}}{d(i)} + \frac{a_{ji}}{d(j)} \Big) + p_{it}$.

Proof.

$$v_t(S) - v_t(S - \{i\}) = \sum_{l \in S} \Big(\sum_{j \in S} \frac{a_{lj}}{d(l)} + p_{lt} \Big) - \Big(\sum_{l \in S - \{i\}} \Big(\sum_{j \in S - \{i\}} \frac{a_{lj}}{d(l)} + p_{lt} \Big) \Big) = \sum_{j \in S - \{i\}} \Big(\frac{a_{ij}}{d(i)} + \frac{a_{ji}}{d(j)} \Big) + p_{it}$$

□

Theorem 1~3 mean that the larger the coalition S is, the higher $v_t(S)$ is.

3.3 Efficient Computation of the Shapley Value for Community Detection

Based on the characteristic function defined in the formula (2), we can compute the Shapley Value for each player to measure its contribution to a coalition. However, the direct computation of the Shapley Value by using the formula (1) is very hard, especially in large networks because the number of combinatorial coalitions is very large. In this section, we develop an iterative formula for computing the Shapley Value to improve the computation efficiency.

Theorem 4. Given a coalitional game model for community detection (N,v), $\forall S_1,S_2\subseteq N$, $S=S_1+S_2$, then the Shapley Value of player $i\in S_1$ and $j\in S_2$ in coalition S with respect to (N,v) can be compute by formula (3):

$$\begin{aligned} SH_t(S,i) &= SH_t(S_1,i)+\frac{1}{2}\sum_{j\in S_2}(\frac{a_{ij}}{d(i)}+\frac{a_{ji}}{d(j)}),\quad i\in S_1 \\ SH_t(S,j) &= SH_t(S_2,j)+\frac{1}{2}\sum_{i\in S_1}(\frac{a_{ij}}{d(i)}+\frac{a_{ji}}{d(j)}),\quad j\in S_2 \end{aligned} \tag{3}$$

Proof: Let $i_1...,i_m,j_1,...,j_n\in N$, $t\in T=\{t_1,t_2,...,t_m\}$, $S_1=\{i_1,...,i_m\}$, $S_2=\{j_1,...,j_n\}$.

(1) Let $S'_2=\{j_1\}$, $S=S_1+S'_2=\{i_1,...,i_m,j_1\}$, then $|S_1|=m$, $|S|=m+1$,

$$\forall i\in S_1,\ SH_t(S,i)=\sum_{\{T'\subseteq S|i\in T\}}\frac{(|T'|-1)!(m+1-|T'|)!}{(m+1)!}[v_t(T')-v_t(T'-\{i\})]$$

$$=\sum_{\{T\subseteq S_1|i\in T\}}\frac{(|T|-1)!(m+1-|T|)!}{(m+1)!}[v_t(T)-v_t(T-\{i\})]$$

$$+\sum_{\{T+\{j_1\}|T\subseteq S_1,i\in T\}}\frac{(|T|)!(m-|T|)!}{(m+1)!}[v_t(T+\{j_1\}\})-v_t(T+\{j_1\}-\{i\})]$$

$$=\sum_{\{T\subseteq S_1|i\in T\}}\frac{(|T|-1)!(m+1-|T|)!}{(m+1)!}[v_t(T)-v_t(T-\{i\})]$$

$$+\sum_{\{T+\{j_1\}|T\subseteq S_1,i\in T\}}\frac{(|T|)!(m-|T|)!}{(m+1)!}[\sum_{j\in T+\{j_1\}-\{i\}}(\frac{a_{ij}}{d(i)}+\frac{a_{ji}}{d(j)})+p_{it}]$$

$$=\sum_{\{T\subseteq S_1|i\in T\}}\frac{(|T|-1)!(m+1-|T|)!}{(m+1)!}[v_t(T)-v_t(T-\{i\})]$$

$$+\sum_{\{T\subseteq S_1|i\in T\}}\frac{(|T|)!(m-|T|)!}{(m+1)!}[\sum_{j\in T-\{i\}}(\frac{a_{ij}}{d(i)}+\frac{a_{ji}}{d(j)})+p_{it}+(\frac{a_{ij_1}}{d(i)}+\frac{a_{j_1i}}{d(j_1)})]$$

$$=\sum_{\{T\subseteq S_1|i\in T\}}\frac{(|T|-1)!(m-|T|)!}{m!}[v_t(T)-v_t(T-\{i\})]$$

$$+\sum_{\{T\subseteq S_1|i\in T\}}\frac{(|T|)!(m-|T|)!}{(m+1)!}[\frac{a_{ij_1}}{d(i)}+\frac{a_{j_1i}}{d(j_1)}]=SH_t(S_1,i)+\frac{1}{2}(\frac{a_{ij_1}}{d(i)}+\frac{a_{j_1i}}{d(j_1)})$$

(2) Let $S_1'=\{i_1,...,i_m,j_1\}=S_1+\{j_1\}, S_2''=\{j_2\}$, then $S=S_1+S_2=\{i_1,...,i_m,j_1,j_2\}$, $|S_1|=m+1$, $|S|=m+2$.

$$SH_t(S,i)=SH_t(S_1')+\frac{1}{2}(\frac{a_{ij_2}}{d(i)}+\frac{a_{j_2i}}{d(j_2)})=SH_t(S_1,i)+\frac{1}{2}\sum_{j\in\{j_1,j_2\}}(\frac{a_{ij}}{d(i)}+\frac{a_{ji}}{d(j)})\cdot$$

……

(3) Let $S_1''=\{i_1,...,i_m,j_1,...,j_{n-1}\}=S_1+\{j_1,...,j_{n-1}\}$, $S_2'''=\{j_n\}$, then $S=S_1+S_2=\{i_1,...,i_m,j_1,...,j_n\}$, $|S_1|=m+n-1$, $|S|=m+n$.

$$SH_t(S,i)=SH_t(S_1'')+\frac{1}{2}(\frac{a_{ij_n}}{d(i)}+\frac{a_{j_n i}}{d(j_n)})=SH_t(S_1,i)+\frac{1}{2}\sum_{j\in\{j_1,...,j_n\}}(\frac{a_{ij}}{d(i)}+\frac{a_{ji}}{d(j)})\ \square$$

$$=SH_t(S_1,i)+\frac{1}{2}\sum_{j\in S_2}(\frac{a_{ij}}{d(i)}+\frac{a_{ji}}{d(j)})$$

The proof for $SH_t(S,j)$, $j\in S_2$ is similar to above, so we do not give the details.

Example 2. Parts of the Shapley Values of players of Fig.1 are given in Tab.2.

Table 2. Parts of the Shapley Values of players of Fig.1 in different coalitions

$SH_{t_1}(\{1\},i)$	$SH_{t_1}(\{2,3\},i)$		$SH_{t_1}(\{1,2,3\},i)$			$SH_{t_1}(\{1,2,3,4\},i)$			
$i=1$	$i=2$	$i=3$	$i=1$	$i=2$	$i=3$	$i=1$	$i=2$	$i=3$	$i=4$
0.5	1.50	1.50	1.33	1.92	1.92	1.66	1.92	1.92	0.83
$SH_{t_2}(\{4\},i)$	$SH_{t_2}(\{5,6\},i)$		$SH_{t_2}(\{4,5,6\},i)$			$SH_{t_2}(\{1,4,5,6\},i)$			
$i=4$	$i=5$	$i=6$	$i=4$	$i=5$	$i=6$	$i=1$	$i=4$	$i=5$	$i=6$
0.5	1.50	1.50	1.33	1.92	1.92	0.83	1.66	1.92	1.92

3.4 The Definition of a Community

In the context of social networks, meaningful communities are groups of individuals that are densely interconnected and have unanimity topics. The feature that individuals are densely interconnected can be revealed based on the structure information of networks, while the feature of unanimity topics can be revealed by the attribute information of individuals. In this paper, the characteristic functions of coalitions defined in the formula (2) incorporate the structure information and the attribute information, and the Shapley Value of each player represents its contribution to the coalition cohesion and topic consistency. If a player can get higher Shapley Value in a coalition than in other coalitions, the player is willing to join this coalition; on the contrary, if a player gets lower Shapley Value in a coalition than in other coalitions, the player will leave this coalition and join another coalition. Therefore, a community can be defined by the member's Shapley Value in a coalition.

Definition 4. A *community.* A community is a coalition in which there is no one member receives lower Shapley Value than that he/she receives from other coalitions, i.e. a coalition S is called a community if $\forall i, SH_t(S,i)\ge SH_{t'}(S',i)$ holds, where $S'\subseteq N, |S'|\le|S|, t,t'\in T$, and t may be same as t'.

Example 3. In the Example 2, coalition $\{1,2,3\}$ is a community in which members closely connected and have the same interested topics (data mining). But coalition $\{1,2,3,4\}$ is not a community, because $SH_{t_1}(\{1,2,3,4\},4) < SH_{t_2}(\{4,5,6\},4)$, it indicates that player 4 would join coalition $\{4,5,6\}$ rather than join coalition $\{1,2,3,4\}$.

3.5 The Algorithm for Detecting Meaningful Communities

Theorem 4 means that if we combine any two small-size coalitions with a same topic to a large-size coalition, then the Shapley Value w.r.t. the large-size coalition will not be lesser than those of small-size coalitions. Therefore, we can expand coalitions by iteratively increasing the size of groups until further increase leads to lower Shapley Values for the members. In this section, we present an algorithm to detect communities by combining coalition level after level. For the first level, players in $N = \{1,2,...,n\}$ form *n* coalitions with respect to a topic. For level *l*, a large-size coalition is formed by adding a player to a coalition of level *l*-1, such combination can produce higher Shapley Value for the new member with respect to the same topic. The combination procedure will stop when the new player gets less Shapley Value from the combined coalition than those it gets from other small-size coalitions. The algorithm is described as follows:

```
Community Detection Algorithm

Input:
$N = \{1,2,...,n\}$, the set of players (individuals)
$A = (a_{ij})_{n \times n}, i, j \in N$, the adjacency matrix
$T = \{t_1, t_2, ..., t_m\}$, the set of finite topics
$P = (p_{it})_{n \times m}, i \in N, t \in T$, the preference matrix
Output: communities
Local variables:
  $l$, a level number
  $r^t$, a coalition number with respect to the topic $t$
  $S_l$, the set of all coalitions in $l$-level
  $S_l^t$, the set of coalitions with respect to the topic $t$ in $l$-level
  $S_{l,r^t}^t$, the $r^t$-th coalition in $S_l^t$
  $Z_i$, the current maximal Shapley Value that player $i$ gets
  $CS$, the set of communities
```

```
Steps:
Step 1. Initialization
 $CS=\Phi$; $l=1$ ; $S_l=\Phi$;
 For $i=1$ to $n$
     $Z_i=0$;
 End for
 For each topic $t\in T$
     $S_l^t=\Phi$;
     For $i=1$ to $n$
         $S_{l,i}^t=\{i\}$; $S_l^t=S_l^t\cup\{S_{l,i}^t\}$; $Z_i=\max(p_{it},Z_i)$;
     End for
     $S_l=S_l\cup S_l^t$;
  End for
Step 2. Form large-size coalitions
Repeat
    $l=l+1$; $S_l=\Phi$ ;
    For each topic $t\in T$
        $r^t=0$; $S_l^t=\Phi$;
    End for
    While $S_{l-1}\neq\Phi$
       $S_{l-1}=\Phi$;
       For each topic $t\in T$
          if $S_{l-1}^t\neq\Phi$
          then
             $(x,y)=\arg\max_{S_{l-1,i}^t\in S_{l-1}^t,\, j\in N-S_{l-1,i}^t}(SH_t(S_{l-1,i}^t\cup S_{1,j}^t,j))$; $y$ is a player whose Shapley Value is maximal in the coalition $S_{l-1,x}^t$
                if $SH_t(S_{l-1,x}^t\cup\{y\},y))>Z_y$
                then
                   $r^t=r^t+1$; $S_{l,r^t}^t=S_{l-1,x}^t\cup\{y\}$; $S_l^t=S_l^t\cup\{S_{l,r^t}^t\}$;
                     $Z_y=SH_t(S_{l,r^t}^t,y),\forall y\in S_{l,r^t}^t$;
                end if
                $S_{l-1}^t=S_{l-1}^t-\{S_{l-1,x}^t\}$;
          end if
            $S_{l-1}=S_{l-1}\cup S_{l-1}^t$;
       end for
       $S_l=S_l\cup S_l^t$;
    end while
Until ($l = n$ or $S_l = \Phi$)
Step 3. generate communities
        $CS = \bigcup_{t \in T} S_l^t$
Step 4. output $CS$
```

The number of the repeat loop is *n*, the most number of while loop is also *n*, there are *m* topics, and the computational complexity for searching *x* and *y* is $O(n^2)$, therefore, the computational complexity of the above algorithm is $O(n \times n \times m \times n^2) = O(mn^4)$. A player may belong to different communities with respect to different topics, and the process for expanding a coalition reveals hierarchical structures of networks.

4 Conclusion

In this paper, we have proposed an approach for detecting meaningful communities in social networks. This approach incorporates the structure information of a network and the attribute information of individuals, by adopting the Shapley Value in cooperative games to measure the preference and the contribution of individuals to a specific topic and to the connection closeness of a coalition. To improve the computation efficiency, an iterative formula for computing the Shapley Value is proposed, and the related theoretical analysis has also been performed. Then a community detection algorithm is developed. We have implemented the algorithms proposed and tests are currently conducted to evaluate various features.

Acknowledgments. This research was supported by the National Natural Science Foundation of China under Grant No.61262069, No.61063008, No.61163003, No.61272126, the Science Foundation of Yunnan Province under Grant No. 2010CD025, and the Yunnan Educational Department Foundation under Grant No.2012C103.

References

1. Ahn, Y.Y., et al.: Link communities reveal multi-scale complexity in networks. Nature 466, 761–764 (2010)
2. Boccaletti, S., et al.: Complex networks: Structure and dynamics. Physics Report 424(4-5), 175–308 (2006)
3. Castro, J., et al.: Polynomial calculation of the Shapley value based on sampling. Computers & Operations Research 36(5), 1726–1730 (2009)
4. Clauset, A., et al.: Hierarchical structure and the prediction of missing links in networks. Nature 453(7191), 98–101 (2008)
5. Fatima, S.S., et al.: A linear approximation method for the Shapley Value. Artificial Intelligence 172, 1673–1699 (2008)

6. Gomez, D., et al.: Centrality and power in social networks: a game theoretic approach. Mathematical Social Sciences 46, 27–54 (2003)
7. Guimera, R., Amaral, L.A.N.: Functional cartography of complex metabolic networks. Nature 433, 895–900 (2005)
8. Han, J., Kamber, M.: Data Mining: Concepts and Techniques, 1st edn. Morgan Kaufmann, California (2000)
9. Ieong, S., Shoham, Y.: Marginal contribution nets: a compact representation scheme for coalitional games. In: Riedl, J., Kearns, M.J., Reiter, M.K. (eds.) Proceedings of the Sixth ACM Conference on Electronic Commerce, EC 2005, Vancouver, BC, Canada, June 5-8, pp. 193–202 (2005)
10. Aadithya, K.V., Ravindran, B., Michalak, T.P., Jennings, N.R.: Efficient computation of the shapley value for centrality in networks. In: Saberi, A. (ed.) WINE 2010. LNCS, vol. 6484, pp. 1–13. Springer, Heidelberg (2010)
11. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell System Technical Journal 49, 291–307 (1970)
12. Kubica, J., et al.: Stochastic link and group detection. In: Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI 2002), Edmonton, Alberta, Canada, July 28-August 1, pp. 798–806. AAAI Press, The MIT Press (2002)
13. Li, D., et al.: Synchronization interfaces and overlapping communities in complex networks. Physics Review Letters 101, 168701 (2008)
14. Lin, Y.R., et al.: Community Discovery via Metagraph Factorization. ACM Transactions on Knowledge Discovery from Data 5(3), 17 (2011)
15. Liu, W.Y., et al.: An approach for multi-objective categorization based on the game theory and Markov process. Applied Soft Computing 11, 4087–4096 (2011)
16. McCallum, A., et al.: Topic and role discovery in social networks. In: Proceedings of the 19th international joint conference on Artificial intelligence, Edinburgh, Scotland, July 30- August 5, pp. 786–791 (2005)
17. Moretti, S., et al.: Using coalitional games on biological networks to measure centrality and power of genes. Bioinformatics 26(21), 2721–2730 (2010)
18. Nash, J.: Non-cooperative games. Annals of Mathematics. Second Series 54(2), 286–295 (1951)
19. Newman, M.E.J.: Fast algorithm for detecting community structure in networks. Physics Review E 69(6), 66133 (2004)
20. Niyato, D., et al.: Controlled Coalitional Games for Cooperative Mobile Social Networks. IEEE Transactions on Vehicular Technology 60(4), 1812–1824 (2011)
21. Owen, G.: Multilinear extensions of games. Management Science 18(5), 64–79 (1972)
22. Palla, G., et al.: Uncovering the overlapping community structure of complex networks in nature and society. Nature 435, 814–818 (2005)
23. Ravasz, E., et al.: Hierarchical organization of modularity in metabolic networks. Science 297, 1551–1555 (2002)
24. Saad, W., et al.: Coalitional game theory for communication networks: a tutorial. IEEE Signal Processing Magazine 26(5), 77–97 (2009)
25. Shapley, L.S.: A Value for N-person games. In: Kuhn, H.W., Tucker, A.W. (eds.) Contributions to be Theory of Games, pp. 307–317. Princeton University Press (1953)
26. Shi, C., et al.: Multi-objective community detection in complex networks. Applied Soft Computing 12, 850–859 (2012)

27. Suri, N.R., Narahari, Y.: Determining the top-k nodes in social networks using the Shapley Value. In: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008), pp. 1509–1512 (2008)
28. Tian, Y., et al.: Efficient aggregation for graph summarization. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, Vancouver, Canada, June 9-12, pp. 567–580 (2008)
29. Wang, F., et al.: Community discovery using nonnegative matrix factorization. Data Mining Knowledge Discovery 22(3), 493–512 (2011)
30. Wang, X.R., et al.: Group and Topic Discovery from Relations and Text. In: Proceedings of the 3rd International Workshop on Link Discovery, LinkKDD 2005, pp. 28–35 (2005)
31. Zhao, Z.Y., et al.: Topic oriented community detection through social objects and link analysis in social networks. Knowledge-Based Systems 26, 164–173 (2012)
32. Zlotkin, G., Rosenschein, J.: Coalition cryptography and stability mechanisms for coalition formation in task oriented domains. In: Association for the Advancement of Artificial Intelligence, pp. 432–437 (1994)

A Learning Classifier-Based Approach to Aligning Data Items and Labels

Neil Anderson and Jun Hong

School of Electronics, Electrical Engineering and Computer Science
Queen's University Belfast, UK
{nanderson423,j.hong}@qub.ac.uk

Abstract. Web databases are now pervasive. Query result pages are dynamically generated from these databases in response to user-submitted queries. A query result page contains a number of data records, each of which consists of data items and their labels. In this paper, we focus on the data alignment problem, in which individual data items and labels from different data records on a query page are aligned into separate columns, each representing a group of semantically similar data items or labels from each of these data records. We present a new approach to the data alignment problem, in which learning classifiers are trained using supervised learning to align data items and labels. Previous approaches to this problem have relied on heuristics and manually-crafted rules, which are difficult to be adapted to new page layouts and designs. In contrast we are motivated to develop learning classifiers which can be easily adapted. We have implemented the proposed learning classifier-based approach in a software prototype, *rAligner*, and our experimental results have shown that the approach is highly effective.

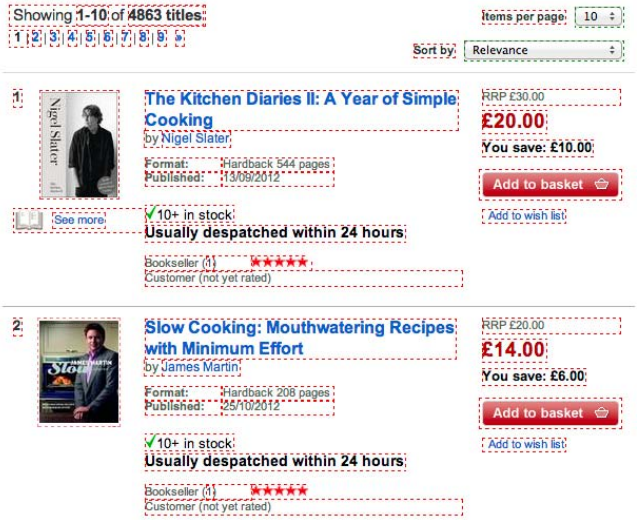
1 Introduction

Web sites that rely on structured databases for their content are ubiquitous. Users retrieve information from these databases by submitting HTML query forms. Query results are displayed on a web page, but in a proprietary presentation format, dictated by the web site designer. We call these pages *query result pages*. Automatic data extraction is the process of extracting automatically a set of data records and the data items that the records contain, from a query result page. Such structured data can then be integrated with data from other data sources and presented to the user in a single cohesive view in response to their query.

Figure 1 illustrates a typical query result page from waterstones.com. On this page each book is presented as a data record, which contains a set of data items and their labels. For example, the book titles, 'The Kitchen...' and 'Slow Cooking...', are examples of data items in each record while 'Format' and 'Published' are examples of labels in each data record. Sometimes, data items and their labels are not separated. For example, in 'RRP £30.00' and 'RRP £20.00' data items and their labels are mixed together. We refer to each of these examples as an *embedded*. So we now consider that a data record contains a set of *data items*, *labels* and *embeddeds*. To ease discussion, we sometimes simply refer to all three of them as *elements* of a data record.

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 282–291, 2013.

In [1] we present a visual approach to data record extraction, which identifies the boundaries of each record on a query result page. The next step of automatic data record extraction from query result pages is to align the elements of each record into different columns and label these columns.



Showing 1-10 of 4863 titles
1 2 3 4 5 6 7 8 9 »
Items per page 10
Sort by Relevance

1
See more
The Kitchen Diaries II: A Year of Simple Cooking
by Nigel Slater
Format: Hardback 544 pages
Published: 13/09/2012
10+ in stock
Usually despatched within 24 hours
Bookseller (1) ★★★★★
Customer (not yet rated)
RRP £30.00
£20.00
You save: £10.00
Add to basket
Add to wish list

2
Slow Cooking: Mouthwatering Recipes with Minimum Effort
by James Martin
Format: Hardback 208 pages
Published: 25/10/2012
10+ in stock
Usually despatched within 24 hours
Bookseller (1) ★★★★★
Customer (not yet rated)
RRP £20.00
£14.00
You save: £6.00
Add to basket
Add to wish list

Fig. 1. A Query Result Page from waterstones.com

Previous approaches to data alignment [9,15] rely on manually-crafted rules. It has been observed that there are design conventions and common patterns across data records in query result pages, which can be represented by manually-crafted rules for data alignment. However, these approaches have a number of limitations.

First, there is no standardised convention for the placement or presentation style of labels or data items on a query result page. The vast range of layout conventions and page designs means that it is difficult to pinpoint precisely which features are important, or how particular features depend on each other. This makes the design of heuristics and rules for alignment very difficult as these tend to end up overly complex in order to deal with the vast range of data items and labels found on query result pages. As a result, it is difficult to design high quality, generalised heuristics or rules that work well across different domains. Second, rules for data alignment are manually crafted on the basis of the query result pages that are available when the rules are created. When page design conventions change, these rules can become obsolete.

In this paper, we propose a learning classifier-based approach to data alignment, which can overcome the limitations described above. Our approach uses two learning classifiers to align data items and labels across data records on query result pages. In this approach, data alignment is done in two phases. In the training phase, a learning classifier is trained with a set of training examples, each of which is represented by a set of automatically extracted features. In the prediction phase, the classifier is used to determine either whether two elements on a query result page are semantically similar, and hence can be grouped together, or whether a group of semantically similar elements are a group of data items, labels or embeddeds.

2 Related Work

Extracting structured data, such as the data found in data records on a query result page, has received a great deal of attention. Early approaches to the problem of data extraction [2,7] aim to build site-specific wrappers to extract data from the corresponding sites. For each wrapper, a set of extraction rules are either proposed by a human expert or automatically induced from a set of labeled examples. Users label examples of everything they want to extract on an example web page from which a wrapper is built. Consequently, data could only be extracted from pages generated using the same template as the example page.

Later approaches to data extraction develop automated algorithms that extract data directly rather than site-specific wrappers. These approaches can be classified by their requirement for domain knowledge.

Domain independent approaches [8,10,12,14], have no requirement for domain knowledge or user intervention in the extraction process. They rely on the appearance of repeated structures in the HTML mark-up or in the visual organisation of a web page. Data items are extracted and aligned by manually specified rules and heuristics based on observations of the repeated structures. These fixed heuristic rules are brittle, even a small change to the design or mark-up of a page can break them. Crucially, this means that these approaches to alignment are predicated on the observations on the query result pages that were available at the time when the approaches were developed. Should the design trends for query result pages change, and then the algorithms must be refactored manually.

In contrast, we are motivated to use a learning-based approach, which is predicated on a number of independent features. Should the design trends change we simply modify or replace our training set of query result pages accordingly. Furthermore, should we discover new features that could be helpful for alignment we can add these to our learning-based approach and rebuild automatically our classifiers to include them. In short, a learning-based approach affords us flexibility that is not achievable when dealing with a fixed set of heuristic rules.

Domain independent data extraction approaches [8,10,12,14] are also vulnerable to noise (such as adverts) appearing in the structure of the web page. To address this limitation, recent *domain dependent* approaches [3,4,5] couple together structural analysis of pages with automatic annotation. The intuition here is that the annotation step identifies the structures on the page that contain the data items, and isolates them from those structures containing noise. This increases extraction accuracy as the algorithms can focus on extracting data, rather than dealing with noise.

Of these, AMBER [5] is the most effective, likely because it tightly couples structural analysis with automated annotations. [3] and [4] both employ a shallower coupling and as a consequence are less able to deal with significant noise. All three approaches rely on some form of domain knowledge or user input to aid the annotation component.

Our approach differs from these three approaches in two ways. First, we rely on our previous work, *rExtractor* [1], to correctly extract each data record (and thus the data items) from a query result page. *rExtractor* is very effective at isolating data from noise on query result pages. Second, our approach is domain independent. Our learning-based classifiers are built automatically from a large training set containing web pages from multiple domains. While it is possible to introduce a degree of domain level tuning (by supplying a training set which contains web pages taken from a single domain) this is not our preference as our approach is designed to negate the need for domain dependence.

3 Learning Classifiers for Aligning Data Items and Labels

The brittle nature of heuristic rules employed by previous data extraction approaches motivated us to consider this problem as a learning task. Given a query result page, we use our previous work *rExtractor* [1] to segment each data record on the page and represent the elements contained as a set of visual blocks. For each block we can extract a number of features, which can be used to 1) characterise the block itself or 2) characterise the relationship between blocks. We would like to learn which of these features are significant for alignment and classification of the block contents.

3.1 Automatically Extracting Features of Visual Blocks

Automatic feature extraction employs a collection of algorithms to extract a feature vector for each visual block. The goal of each algorithm is to extract a particular type of feature which represents the visual, structural or content characteristics of the block.

Visual Features. The visual appearance of each visual block displayed on the query result page is determined by its visual properties. There are 160 individual visual properties that can be applied to each visual block by the web page designer. These range from properties such as font size and colour, to more obscure properties such as opacity.

The intuition is that the designer uses the same visual appearance for each visual block in the same group of elements in each data record. For instance, as shown in Figure 1, the visual block that contains the book title has the same visual appearance in both data records. This observation is likely to hold because data records on query result pages are generated by template. There are cases where there are differences between some of the visual properties of two visual blocks in the same similarity group. For example, some designers use alternating background colours to create a visual distinction between rows of data records. Accordingly, our approach considers each of the 160 visual properties as an independent feature. It is left up to the classifier to determine which of the visual properties characterises the similarity between two visual blocks.

Identity Features. Each visual block is rendered from a node in the Document Object Model (DOM) of the query result page. Typically, such a node has an ID, node name and class name assigned to it, each of which is a text string. We call each of these an *identity feature*. For example, as shown in Figure 1, the visual block that contains the label ‘Format’ has been assigned the ID ‘productFormat’, while the visual block that contains the label ‘Published’ has been assigned the ID ‘datePublished’.

The intuition is that the designer uses the same identify features for visual blocks in the same similarity group. Again, this observation is likely to hold because data records on query result pages are generated by template. In this example, as the visual blocks which contain the two different groups of labels ‘Format’ and ‘Published’ have the same visual appearance, their identify features become the discriminating features for alignment.

Visual Block Content Similarity Feature. This feature characterises whether two visual blocks contain similar contents. The intuition is that if two visual blocks contain similar contents, they should be probably aligned into the same similarity group. For example, a visual block from one record may contain the text ‘You Save: £6.00’ and another visual block from another record may contain the text ‘You Save: £10.00’. Both blocks contain very similar contents and therefore should be aligned into the same similarity group. Our approach uses the widely used cosine coefficient [11] which is a effective technique to compute numeric similarity between the string contents of two visual blocks.

Definition 1. *For two visual blocks A and B, let $A_f = (A_1, A_2, ..., A_n)$ and $B_f = (B_1, B_2, ..., B_n)$ be the term frequency vectors of the characters contained in the string contents of A and B respectively. The cosine similarity between A and B is defined as follows:*

$$cosineSimilarity(A, B) = \frac{A_f \cdot B_f}{\|A_f\| \, \|B_f\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

A real value ranging between 0 and 1 represents the cosine similarity between the string contents of A and B with 1 meaning the exactly the same and 0 indicating the opposite, and in-between values indicating an intermediate degree of similarity.

Formatting Features. Text styles and decorations are often good indicators of what a visual block contains, that is, whether it contains a label, data item or embedded. For instance, a visual block that contains an underlined text may be a label. The full set of text properties for a visual block are used as formatting features, including colour, direction, letter-spacing, line-height, text-align, text-decoration, text-indent, text-shadow, text-transform, vertical-align, white-space and word-spacing.

Punctuation Features. Separators are often good indicators of whether a visual block contains an embedded. For instance, a visual block that contains a separator such as a colon may be an embedded. For example, as shown in Figure 1, the first record contains a visual block with text contents ‘You Save: £10.00’. This visual block is an embedded, that is, it is a combination of a label and a data item. In this example, the colon acts as a separator between the label, ‘You Save’, and the data item, ‘£10.00’. Given a similarity group, these features are used to represent whether a set of separators appear in the contents of a representative visual block from the group. The contents of the representative visual block are scanned against a preset list of separators, which have been selected as they are commonly used by designers to delimit a label and a data item. The selected separators are $\{;, :, |, -\}$. Each separator is represented as a binary feature, with the value of 1 for the appearance of the separator and the value of 0 otherwise.

Longest Common Substring Features. The presence of the same substring in the contents of each visual block in a similarity group is another indicator of whether the group contains a set of embeddeds. For instance, a similarity group that has a common substring may indicate that the group contains a group of embeddeds which share the same label but contain different data items. Our approach only considers substrings that appear as common prefixes or suffixes of the contents of the visual blocks.

3.2 Learning Classifiers

Our approach to aligning labels and data items on a query result page employs two learning classifiers. The first classifier is for determining whether two visual blocks are semantically similar and hence should be aligned into the same similarity group. The second classifier is for determining the type of a similarity group, that is, whether it is a group of labels, data items or embeddeds.

Creating Feature Vectors. For the first classifier, a feature vector V_1 is created for a pair of visual blocks, A and B. The first features in V_1 represent whether each pair of the corresponding visual features in A and B match. The remaining features in V_1 represent whether the corresponding identity features of A and B match respectively and the visual block content similarity between A and B.

For the second classifier, a feature vector V_2 is created for each similarity group. A representative visual block is chosen from the group. The formatting and punctuation features of the representative block represents the corresponding features of the group, which are included in the feature vector. In addition, the two longest common substring features of the group are also included in the feature vector for the group.

Training Classifiers. In order to train the first learning classifier, we create a training dataset, T_1, which contains a certain number of pairs of visual blocks between two data records on the same query result page. We first choose two records from the query result page with the top two numbers of visual blocks. We then pair up the visual blocks, each from one of the two chosen data records to create a set of visual block pairs. For each pair of visual blocks, a feature vector is created, and the pair is manually tagged as either positive or negative, depending whether or not they are semantically similar. Each training example is represented as a triple of the form:

$$((A, B), V_1, L)$$

where (A, B) represents a pair of visual blocks, V_1 is the feature vector for the pair, and L is the manually tagged class label.

To train the second classifier, we create a training dataset, T_2, which contain a certain number of similarity groups of visual blocks. Each similarity group is manually tagged as containing labels, data items or embeddeds based on their display on the rendered query result page. Each training example is represented as a triple of the form:

$$(G, V_2, L)$$

where G represents a similarity group of visual blocks, V_2 is the feature vector for the group, and L is the manually tagged group type label.

4 Using Classifiers to Align Data Items and Labels

In the alignment stage, for a given query result page, we use two trained classifiers: one is for clustering each visual block that represents a data item, label or embedded into a similarity group, and another is for determining the type of each similarity group.

4.1 Using First Classifier as Similarity Function

We use the first classifier as a similarity function, $Similarity(A, B)$, which takes a pair of visual blocks, A and B, as input and determines whether they are semantically similar. Each pair of visual blocks is represented by a feature vector which is taken as input by the first classifier and the classifier produces as output a Boolean value to indicate whether two visual blocks are semantically similar.

4.2 Aligning Data Items and Labels

Our approach implements a single-pass, clustering-based approach to alignment. As shown in Algorithm 1, each visual block in each record is aligned into a strict partition, which uses the first classifier as similarity function. The algorithm processes all of the visual blocks in the first record, followed by the second record and so on (lines 3 and 4). For each visual block the algorithm decides whether it is similar to a visual block in one of the existing clusters (line 6). If so, the block is added to the cluster (line 7); Otherwise, a new cluster is created to contain the visual block itself (line 9). Finally, a set of clusters, S, is returned (line 10).

Algorithm 1. Alignment Algorithm

```
 1: Input: a set of data records, $R$, from a query result page.
 2: Output: clusters of aligned visual blocks, $S$.
 3: for all $r \in R$ do
 4:    for each visual block $b \in r$ do
 5:       for each cluster $s \in S$ do
 6:          if there exits $b' \in s$ such that $Similarity(b, b')$ then
 7:             $s \leftarrow s \cup \{b\}$
 8:          else
 9:             $S \leftarrow S \cup \{b\}$
10: return $S$
```

5 Experimental Evaluation

In order to select the best learning technique to build each classifier, we use two datasets to evaluate experimentally a number of learning techniques. For each classifier, we perform a 10-fold cross-validation on each technique to assess how well it performs.

5.1 Datasets

We use two datasets, *DS1* and *DS2*. DS1 is used to evaluate and select a learning technique for the first classifier. DS2, derived from the results of the first classifier, is used to evaluate and select a learning technique for the second classifier. As there is no standard dataset for analysis of data record alignment techniques, we have created these datasets. DS1 comprises of 7,600 classification instances derived from query result pages from 51 distinct web sites taken from the third-party testbed presented in [13] in addition to 149 query result pages taken from the testbed used in [1].

Table 1. Results for First Classifier

Technique	Recall	Precision	F-Measure
J48 Decision Tree	0.970	0.971	0.97
Logistic	0.970	0.971	0.97
IB1	0.970	0.970	0.97
SVM	0.967	0.970	0.969
Naïve Bayes	0.927	0.952	0.933

Table 2. Results for Second Classifier

Technique	Recall	Precision	F-Measure
Naïve Bayes	0.877	0.88	0.877
Logistic	0.867	0.865	0.865
IB1	0.854	0.854	0.854
SVM	0.83	0.837	0.82
J48 Decision Tree	0.813	0.818	0.813

5.2 Selection of Learning Techniques

We use WEKA [6] to evaluate the performance of different learning techniques. For each classifier, we considered: J48 Decision Tree, Logistic, IB1, SVM and Naïve Bayes. In Tables 1 and 2 we present the recall, precision and f-measure (exclusive of errors made by *rExtractor*) for each learning technique on both classifiers.

Selection of First Classifier. As shown in Table 1, the J48 Decision Tree, Logistic, IB1 and SVM learning techniques all exhibit uniformly high weighted average scores for recall, precision and f-measure for the first classifier. Only by inspecting the raw statistics of correctly and incorrectly classified instances are we able to determine that J48 Decision Tree is statistically the most effective learning technique for the first classifier as it has the highest number of correctly classified instances.

As Table 1 shows, Naïve Bayes is the least effective learning technique for the first classifier. A limitation of the Naïve Bayes technique may explain this disparity of performance. Naïve Bayes assumes that all of the features are independent, which for the first classifier is not always the case. For instance, making an assessment based on a feature such as a visual property match or an identity feature match in isolation is not always enough to determine if two blocks should be aligned. In contrast, decision trees are an effective technique when dealing with correlated features.

Run-time analysis of each algorithm reveals that Naïve Bayes is the most efficient algorithm followed by J48 which executes 4.5 times faster than Logistic, 15 times faster than IB1 and 2 times faster than SVM.

Detailed Analysis of First Classifier. The failure to align two blocks from two different data records, which are semantically similar, into the same similarity group is the most common error made by the first classifier. This can occur, for instance, if each data record contains two blocks that have the same visual properties and identity features. In this case, the block content similarity becomes the only remaining discriminating feature for alignment. This may still work if two blocks from two different data records, which are semantically similar, have high block content similarity. However, if the block content similarity turns out too low, the classifier would fail to determine if the blocks are semantically similar. For example, two blocks from two data records containing two dates'13/09/2012' and '22/10/2012' have low block content similarity and hence would not be aligned into the same similarity group even though they both represent dates.

Selection of Second Classifier. As shown in Table 2, Naïve Bayes is the most effective learning technique for the second classifier. Interestingly, unlike the first classifier, the J48 Decision Tree is the least effective learning technique, for two reasons. First, there is no single feature that can be used by the second classifier to split effectively the outcome of the tree. Second, there are also fewer correlated features, therefore, a multi-layer decision tree, such as the J48 Decision Tree, which is effective at modelling correlated features, is less effective. For this particular classifier, it is much better to consider all of the features independently, a task to which Naïve Bayes is ideally suited.

Detailed Analysis of Second Classifier. The most common error made by the second classifier is the misclassification of a group of data items as a group of embeddeds. This happens when the string contents of the data items each contain a long common substring, but do not completely match. If these conditions are met, the classifier incorrectly assumes that the data item is composed of a label and a data item, and therefore assigns the embedded classification type to the group. This arrangement can happen legitimately on a query result page. For example, a user searching for a manufacturer, such as 'Dyson' could see results such as: 'Dyson DC27', 'Dyson DC40', 'Dyson DC24'.

6 Conclusions

This paper presents a novel, machine learning-based, approach to the automatic alignment of the contents of data records from a query result page. Our approach first automatically extracts a number of features which characterise the contents of the data records. Next, we create two feature vectors which are used to build two classifiers. The first classifier aligns the elements of the data records into similarity groups, while the second classifies each similarity group. Finally, for each classifier we evaluate experimentally a number of competing learning techniques and, on the basis of the experimental results, select the most effective learning technique for each classifier. Our experimental results show that the proposed learning techniques are highly effective.

References

1. Anderson, N., Hong, J.: Visually extracting data records from query result pages. In: Ishikawa, Y., Li, J., Wang, W., Zhang, R., Zhang, W. (eds.) APWeb 2013. LNCS, vol. 7808, pp. 392–403. Springer, Heidelberg (2013)
2. Baumgartner, R., Flesca, S., Gottlob, G.: Visual web information extraction with lixto. The VLDB Journal, 119–128 (2001)
3. Dalvi, N., Kumar, R., Soliman, M.: Automatic wrappers for large scale web extraction. Proc. VLDB Endow. 4(4), 219–230 (2011)
4. Derouiche, N., Cautis, B., Abdessalem, T.: Automatic extraction of structured web data with domain knowledge. In: ICDE, Washington, DC, USA, pp. 726–737 (2012)
5. Furche, T., Gottlob, G., Grasso, G., Orsi, G., Schallhart, C., Wang, C.: Little knowledge rules the web: Domain-centric result page extraction. In: Rudolph, S., Gutierrez, C. (eds.) RR 2011. LNCS, vol. 6902, pp. 61–76. Springer, Heidelberg (2011)
6. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. SIGKDD 11(1), 10–18 (2009)
7. Kushmerick, N.: Wrapper induction for information extraction. PhD thesis (1997)
8. Liu, W., Meng, X., Meng, W.: Vide: A vision-based approach for deep web data extraction. IEEE Transactions on Knowledge and Data Engineering 22, 447–460 (2010)
9. Lu, Y., He, H., Meng, W., Zhao, H., Yu, C.: Annotating structured data of the deep web. In: 23rd Conf. on Data Engineering, pp. 376–385. Society Press (2007)
10. Simon, K., Lausen, G.: Viper: augmenting automatic information extraction with visual perceptions. In: CIKM Conference, New York, NY, USA, pp. 381–388 (2005)
11. Singhal, A.: Modern information retrieval: a brief overview. A bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (2001)
12. Wang, J., Lochovsky, F.H.: Data extraction and label assignment for web databases. In: WWW Conference, New York, NY, USA, pp. 187–196 (2003)
13. Yamada, Y., Craswell, N., Nakatoh, T., Hirokawa, S.: Testbed for information extraction from deep web. In: WWW Conference, New York, pp. 346–347 (2004)
14. Zhai, Y., Liu, B.: Web data extraction based on partial tree alignment. In: WWW Conference, New York, NY, USA, pp. 76–85 (2005)
15. Zhao, H., Meng, W., Wu, Z., Raghavan, V., Yu, C.: Fully automatic wrapper generation for search engines. In: WWW Conference, pp. 66–75 (2005)

Self-supervised Automated Wrapper Generation for Weblog Data Extraction

George Gkotsis, Karen Stepanyan, Alexandra I. Cristea, and Mike Joy

Department of Computer Science,
University of Warwick,
Coventry CV4 7AL,
United Kingdom
{G.Gkotsis,K.Stepanyan,A.I.Cristea,M.S.Joy}@warwick.ac.uk

Abstract. Data extraction from the web is notoriously hard. Of the types of resources available on the web, weblogs are becoming increasingly important due to the continued growth of the blogosphere, but remain poorly explored. Past approaches to data extraction from weblogs have often involved manual intervention and suffer from low scalability. This paper proposes a fully automated information extraction methodology based on the use of web feeds and processing of HTML. The approach includes a model for generating a wrapper that exploits web feeds for deriving a set of extraction rules automatically. Instead of performing a pairwise comparison between posts, the model matches the values of the web feeds against their corresponding HTML elements retrieved from multiple weblog posts. It adopts a probabilistic approach for deriving a set of rules and automating the process of wrapper generation. An evaluation of the model is conducted on a dataset of 2,393 posts and the results (92% accuracy) show that the proposed technique enables robust extraction of weblog properties and can be applied across the blogosphere for applications such as improved information retrieval and more robust web preservation initiatives.

Keywords: Web Information Extraction, Automatic Wrapper Induction, Weblogs.

1 Introduction

The problem of web information extraction dates back to the early days of the web and is fascinating and genuinely hard. The web, and the blogosphere as a constituent part, correspond to a massive, publicly accessible unstructured data source. Although exact numbers of weblogs are not known, it is evident that the size of the blogosphere is large. In 2008 alone the Internet company Technorati reported to be tracking more than 112 million weblogs, with around 900 thousand blog posts added every 24 hours[1]. In Britain alone, 25% of Internet

[1] http://technorati.com/blogging/article/state-of-the-blogosphere-introduction/

G. Gottlob et al. (Eds.): BNCOD 2013, LNCS 7968, pp. 292–302, 2013.

users maintain weblogs or personal websites [5] that are read by estimated 77% of Web users. Hence, the volume of information published on weblogs justifies the attention of information retrieval, preservation and socio-historical research communities.

The scale is not the only challenge for capturing weblog resources. The heterogeneous nature of these resources, the large numbers of third party elements and advertisements, the rapid changes, the propagation of user-generated content and the diversity of inter-relations across the resources are among the common characteristics of the blogosphere. These characteristics amplify the complexity of capturing, processing and transforming these web resources into structured data. The successful extraction of weblog properties is of paramount importance for improving the quality of numerous applications, such as analytics, information retrieval and preservation.

Typically, the content of a weblog resides in a relational database. The automation supported by the blogging platform provides a common structure that can be observed across the various weblog pages. More specifically, the weblog post, which constitutes a building block of a weblog, comprises a set of properties, such as the title, author, publication date, post content and the categories (or tags) assigned. Whilst the data structure is presented in a consistent way across a certain weblog, it rarely appears identical across different weblogs, even if the blogging platform remains the same. The main reason for the above inconsistency is the fact that bloggers personalise the presentation of their weblog arbitrarily, hence the resulting weblog exhibits a unique appearance. Moreover, current techniques are not sufficient to meet the requirements of a weblog data extraction framework which is a)fully automated, b)high granularity and c)high quality.

One of the most prominent characteristics of weblogs is the existence of web feeds. Web feeds, commonly provided as RSS, are machine interpretable, structured XML documents that allow access to (part of) the content of a website, such as a weblog. In fact, this high quality, rigorous information contained in web feeds has already been exploited in several applications (e.g. [13]). The solution proposed here is influenced by the above idea of exploiting the web feeds and attempts to overcome the limitation of fixed number of provided post-entities. Intuitively, our approach is not to treat the web feeds as the only source of information, but as a means that allows the *self-supervised* training and generation of a wrapper automatically. During this self-supervised training session[2], the matching of the elements found between the web feeds and the weblog posts is used to observe and record the position of the cross-matched elements. Based on these observations, a set of rules is generated through an essentially probabilistic approach. These rules are later applied throughout each weblog post (regardless of its existence in the web feed).

[2] The term self-supervised is inspired by and used in a similar way by Yates et al. in order to describe their classifier induction [17]. Contrary to our approach, where we focus on data values, their approach concerns the extraction of relational information found in texts without using web feeds.

This research makes the following main contributions:

- We use web feeds for training and generating a wrapper. The generated wrapper is described in simple rules that are induced by following a probabilistic approach. We provide a simple algorithm that is noise-tolerant and takes into account the information collected about the location of HTML elements found during training.
- We make use of CSS Classes as an attribute that can supplement the more traditional XPath manipulation approach used to describe extraction rules.
- To the best of our knowledge, we are the first to propose a self-supervised methodology that can be applied on any weblog and features unprecedented levels of granularity, automation and accuracy. We support all of the above through evaluation.

The paper is structured as follows. Section 2 describes the proposed model and the methodology applied to extract the desired weblog properties. Section 3 evaluates the model, while Section 4 discusses the contribution of the approach and presents related work. Finally, Section 5 presents the conclusions.

2 Proposed Model

We adopt the definition of a wrapper proposed by Baumgartner et al. where “a wrapper is a program that identifies the desired data on target pages, extracts the data and transforms it into a structured format” [3]. As discussed above, our model aims to generate a fully automated wrapper for each weblog. The approach is divided into three steps as follows.

2.1 Step 1: Feed Processing and Capturing of Post Properties

The input for executing the first step of the proposed model involves the acquisition of the desired blog’s feed content. Similarly to standard RSS readers, the model focuses on the entries that point to the weblog posts. For each entry, we search and store the attributes of *title*, *author*, *date published*, *summary* and *permalink* as the post properties.

2.2 Step 2: Generation of Filters

The second step includes the generation of filters. The naming convention we use for the concept of a *filter* is similar to the one introduced in [2], where it is described as the building block of patterns, which in turn describes a generalised tree path in the HTML parse tree. Thus, adding a filter to a pattern extends the set of extracted targets, whereas imposing a condition on a filter restricts the set of targets. The same concept is used by XWRAP [11] in order to describe the so-called “declarative information extraction rules”. These rules are described in XPath-like expressions and point to regions of the HTML document that contain data records.

Following related work, we use the concept of a filter in order to identify and describe specific data elements of an HTML weblog post. Unlike previous work, where most of the tools deal with the absolute path only (for example through partial tree alignment [18]), our filters comprise a tuple, which extends existing approaches. Our approach overcomes irregularities appearing across absolute path values by providing additional, alternate means of describing the HTML element (namely our tuple also includes CSS Classes and HTML IDs). By conducting an initial visual survey on weblogs, we hypothesize that especially CSS Classes may be used to provide an alternate and accurate way to induce extraction rules, a feature that remains unexploited in most (if not all) approaches until now. Our evaluation results support the above hypothesis.

In our approach, the filter is described using three basic attributes, as follows.

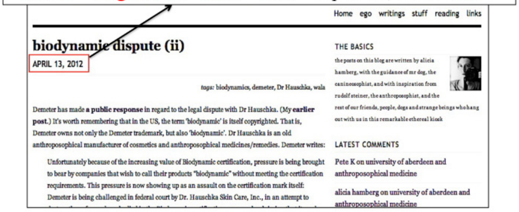
- Absolute Path: We use a notation similar to XPath's absolute path to refer to the position of the HTML element. The Absolute Path is described as a sequence of edges, where each edge is defined as the *name* of the element and the positional information of the element (*index*)[3]. This sequence of edges starts from the root of the document and ends with the element containing the value we are interested in. For example, the value */html[0]/body[1]* refers to the body element of an HTML document, since this is the second child (hence, *body[1]*) of the root HTML element (*html[0]*).
- CSS Classes: "CSS (Cascading Style Sheets) is a simple mechanism for adding style (e.g., fonts, colours, spacing) to web documents"[4], first introduced in 1996. It allows the separation of document content from document presentation through the definition of a set of rules.
- HTML ID: The ID attribute specifies a unique identifier for an HTML element of a document. It is commonly used in cases where CSS code needs to refer to one unique element (e.g. the title of a post) or run JavaScript.

Figure 1 shows the structure of a filter with an annotated example. When pointing at a specific element, a set of HTML ID values and CSS Classes together with a single-valued Absolute Path are used to describe and define the filter. More specifically, when an element is identified, any HTML IDs or CSS Classes applied on this element are added to the filter. Afterwards, an iterative selection of the parent element continues, adding HTML IDs and CSS Classes to the sets, as long as the value of the parent element contains nothing but the value identified. For the example illustrated in Figure 1, the value for the ID attribute is *single-date*, for the CSS Classes the value is *date* and the Absolute Path is *html[0]/body[1]/div[1]/div[1]/div[0]/div[0]/div[1]*.

[3] The positional information of an HTML element is crucial in a HTML document. This is one of the reasons that HTML DOM trees are viewed as labelled ordered trees in the literature (e.g., [7]).

[4] `http://www.w3.org/Style/CSS/`

<div id="single-date" class="date">April 13, 2012</div>



Home ego writings stuff reading links

biodynamic dispute (ii)

APRIL 13, 2012

tags: biodynamics, demeter, Dr Hauschka, wala

Demeter has made **a public response** in regard to the legal dispute with Dr Hauschka. (My **earlier post.**) It's worth remembering that in the US, the term 'biodynamic' is itself copyrighted. That is, Demeter owns not only the Demeter trademark, but also 'biodynamic'. Dr Hauschka is an old anthroposophical manufacturer of cosmetics and anthroposophical medicines/remedies. Demeter writes:

> Unfortunately because of the increasing value of Biodynamic certification, pressure is being brought to bear by companies that wish to call their products "biodynamic" without meeting the certification requirements. This pressure is now showing up as an assault on the certification mark itself: Demeter is being challenged in federal court by Dr. Hauschka Skin Care, Inc., in an attempt to

THE BASICS

the posts on this blog are written by alicia hamberg, with the guidance of mr dog, the canineosophist, and with inspiration from rudolf steiner, the anthroposophist, and the rest of our friends, people, dogs and strange beings who hang out with us in this remarkable ethereal kiosk

LATEST COMMENTS

Pete K on university of aberdeen and anthroposophical medicine

alicia hamberg on university of aberdeen and anthroposophical medicine

IDs	CSS Classes	Absolute Path
single-date	date	html[0]/body[1]/div[1]/div[1]/div[0]/div[0]/div[1]

Fig. 1. The structure of a filter. An example is annotated for the case of an element containing a date value.

2.3 Step 3: Induction of Rules and Blog Data Extraction

After the completion of Step 2, a collection of filters is generated for each property. When applied to the weblog posts from which they were extracted, they link back to the HTML element of the matched property. However, due to multiple occurrences of values during the text matching process of Step 2, there are cases where a value is found in more than one HTML element. This results in generating filters equal to the number of values found. Not all of the collected filters are suitable for extracting the data out of a weblog; the collection contains diverse and sometimes "blurred" information that needs further processing.

In the case of weblog data extraction, there is neither prior knowledge of the location of the elements to be identified, nor a definite, automated way to describe them. We propose a case-based reasoning mechanism that assesses the information found in filters. The aim of this mechanism is to generate rules through a *learning by example* methodology, i.e., a general *rule* is extracted through the observation of a set of *instances*. In our case, the *instances* correspond to the weblog posts that lead to the generation of the filters during the previous step. The *rules* are defined in the language used to describe the previously collected filters. Therefore, they describe how to extract the weblog properties. Our approach deals with irregularities found in web documents (and filters thereof) in an inherently probabilistic way.

During the step of the induction of rules, our aim is to account for each attribute of each filter (Absolute Path, CSS Classes and HTML IDs), in order to assess the usefuleness of each attribute value as a rule. Essentially, a rule is the result of the transposition of a filter. This transposition can result in maximum three rules. The rule is described by its *type* (one of the three different attribute types of the filters), a *value* (the value of the corresponding filter's attribute) and a *score*, which is used to measure its expected accuracy. An important consideration taken here is the fact that selecting a "best-match" filter (i.e. a tuple of *all* three filter attributes) from the list of the filters or a combination of values for *each* attribute (as a collection of "best-of" values for each attribute) may result in the elimination of the desired element (for example, the HTML ID might increment for each weblog post and is therefore unique for every instance). The approach adopted here is based on the assumption that a single attribute (either the Absolute Path, or the CSS Classes or the HTML IDs) should suffice for describing the extraction rule. In the evaluation section, we give evidence why using a single attribute meets this expectation.

Algorithm 1. Rule induction algorithm

```
Inputs:
    Collection of training posts $P$, Collection of candidate rules $R$
Outputs:
    Rule with the highest score
for all Rules $r \in R$ do                                  ▷ Initialize all scores
    $r.score \leftarrow 0$
end for
Rule $rs \leftarrow$ new Rule()
$rs.score \leftarrow 0$
for all Rules $r \in R$ do
    for all Posts $p \in P$ do  ▷ Check if application $r(p)$ of rule $r$, on post $p$ succeeds
        If $r(p)$ =value-property of $p$ then
            $r.score++$
    end for
    $r.score \leftarrow \frac{r.score}{|P|}$                       ▷ Normalize score values
    If $r.score > rs.score$ then                 ▷ Check if this is the best rule so far
        $rs \leftarrow r$
end for
return $rs$
```

To assess the rule that best describes the extraction process, a score is calculated for each rule. The score aims at keeping track of the effectiveness of the rule, when applied across different posts of the weblog. As seen in Algorithm 1, an iteration takes place for each of the candidate rules, which in turn is applied on each of the training posts. For each successful match, the score of the rule is increased by one[5]. After all posts have been checked, the value is divided by the number of training posts against which the rule was validated, in order to represent a more meaningful, normalised measurement (i.e. the higher the better: 1 means that rule is successful for all posts, 0 means that it failed for each of the posts applied). The rule having the highest score – if any – is returned.

Figure 2 presents an overview of the approach described above. As already discussed in detail, the proposed solution involves the execution of three steps. The first step includes the task of reading and storing the weblog properties found in the web feed. The second step includes training the wrapper through the cross matching of information found in the web feed and the corresponding

[5] A successful match between properties is a crucial issue in our approach and is discussed in detail in Section 2.4.

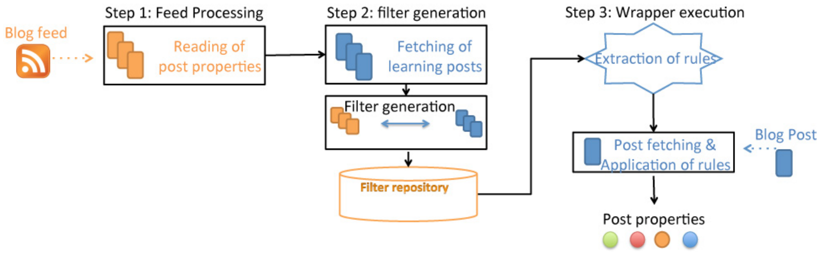

Step 1: Feed Processing
Step 2: filter generation
Step 3: Wrapper execution
Blog feed
Reading of post properties
Fetching of learning posts
Extraction of rules
Filter generation
Filter repository
Post fetching & Application of rules
Blog Post
Post properties

Fig. 2. Overview of the weblog data extraction approach

HTML documents. This step leads to the generation of information, captured through the filters, which describes where the weblog data properties reside. The final step transforms the filters into rules and calculates the rule scores in order to select a rule for each of the desired properties.

2.4 Property Matching

The proposed method relies on the identification of an HTML element against a specific value. Text matching can be used for achieving the above identification. Generally, text matching is not a trivial task and can be classified into various string matching types. More specifically, text matching may be complete, partial, absolute or approximate. The matching of the elements is treated differently for different properties, which is another contribution of this paper (details have been omitted due to space limitations). For the title we look for absolute and complete matchings, for the content we use the Jaro-Winkler metric [15] which returns high similarity values when comparing the summary (feed) against the actual content (web page), for the date we use the Stanford NER suite for spotting and parsing the values [6], and for the author we use partial and absolute matching with some boilerplate text (i.e. "Written By" and "Posted By").

3 Evaluation

We evaluated our model against a collection of 240 weblogs (2,393 posts) for the title, author, content and publication date. For the same collection, we used the Google Blogger and WordPress APIs (in the limits of free quota) in order to get valid and full data (i.e. full post content) and followed the 10-fold validation technique [16]. As seen in Table 1, the prediction accuracy is high (mean value *92%*). For the case of the title, the accuracy is as high as *97.3%* (65 misses). For the case of the content, the accuracy is *95.9%* (99 misses). Publication date is 89.4% accurate (253 misses) and post author is 85.4% (264 misses). Table 1 summarizes the above results and presents the accuracy of Boilerpipe (77.4%) [8] (Boilerpipe is presented in detail in Section 4). Concerning the extraction of the title using Boilerpipe, the captured values are considered wrong, since the tool

Table 1. Results of the evaluation showing the percentage of successfully extracted properties. Number of misses are in parenthesis.

	Title	Content	Publication Date	Author
Proposed Model	97.3%(65)	95.9% (99)	89.4% (253)	85.4% (264)
Boilerpipe	0	77.4% (539)	N/A	N/A

extracts the title of the HTML document. For the case of the main content, our model achieves *81.6%* relative error reduction. Furthermore, the overall average score for all rules is *0.89*, which presumably indicates that the induction of the selected rules is taking place at a high confidence level.

4 Discussion and Related Work

The concept of using web feeds for capturing data is not new. ArchivePress is one of the weblog archiving projects that have developed solutions for harvesting the content of weblog feeds [14]. The solution focuses solely on collecting the structured content of weblog feeds that contain posts, comments and embedded media. The solution provided by ArchivePress remains highly limited, due to the fixed number of entries and partial content (i.e. post summary) found in feeds. Another approach that attempts to exploit web feeds was developed by Oita and Sellenart [13]. This approach is based on evaluating a web page and matching it to the properties found in the corresponding web feed. The general principle of cross-matching web feeds and pages constitutes the foundation of the approach that we propose in this paper. However, because the approach by Oita and Sellenart does not devise general extraction rules, it remains inapplicable for capturing the data that are no longer available in the corresponding web feed. Additionally, the performance of their approach for extracting distinct properties such as title was reported as poor (no figures were provided in the paper).

To position our approach from a more general point of view (within the domain of earlier conducted work on web information extraction), we classify it according to the taxonomy of data extraction tools by Laender [10]. More specifically, our approach can be associated with the Wrapper Induction and Modelling-Based approaches. Similarly to the wrapper induction tools, our approach generates extraction rules. However, unlike many wrapper induction tools, our approach is fully automated and does not rely on human-annotated examples. Instead, it uses web feeds as a model that informs the process of generating extraction rules and it therefore resembles the Modelling-Based approaches. Hence, the approach presented in this paper can be positioned in relation to tools such as WIEN [9], Stalker [12], RoadRunner [4] or NoDoSE [1].

WIEN is among the first tools aimed at automating the process of information extraction from web resources. The term *wrapper induction* is, in fact, coined by the authors [9] of the tool. However, as one of the earlier attempts, the use of the tool is restricted to a specific structure of the page and the heuristics of the presented data. Furthermore, it is not designed to work with nested structure of web data. The limitation of working with hierarchical data has been addressed by the Stalker tool [12]. However, the use of the Stalker tool requires a supervised training data set that limits the degree of automation offered by the system. An attempt to automate the process of wrapper induction was made by Crescenzi et al. [4] and published along with the RoadRunner tool. The tool analyses structurally pairs of similar resources and infers an unlabelled (i.e. no property identified) schema for extracting the data. NoDoSE [1] represents a different, modelling-based category of tools that requires an existing model that defines the process of extraction. This is a semi-automatic approach due to the necessary human input for developing models. However, additional tools, such as a graphical user interface for marking resources, can be used for facilitating human input. Hence, the review of the earlier work suggests that our approach, as proposed in this paper, addresses a niche not served by the existing tools.

Among the generic solutions there are other technologies that aim at identifying the main section (e.g. article) of a web page. The open source Boilerpipe system is state-of-the-art and one of the most prominent tools for analysing the content of a web page [8]. Boilerpipe makes use of the structural features, such as HTML tags or sequences of tags forming subtrees, and employs methods that stem from quantitative linguistics. Using measures, such as average word length and average sentence length, Boilerpipe analyses the content of each web page segment and identifies the main section by selecting the candidate with the highest score. As reported by Oita and Sellenart [13], the use of Boilerpipe delivers relatively good precision (62.5%), but not as high as our approach.

Lastly, it is necessary to discuss the limitations of the proposed model and future work. First of all, a requirement for the adoption of the model is the existence and integrity of web feeds. While web feeds are prominent characteristics of weblogs, some weblogs are not configured to publish their updates through feeds. In this case, the proposed model would not be appropriate to extract any data. Additionally, a technique used to deceive anti-spam services is to report false information in web feeds. In this case, the proposed model will signal a low score on the rules – if any – generated (in fact, this limitation may be further considered for spam detection). Another limitation concerns the extraction of the *date* property. The date is currently processed for the English language only, which may pose problems when matching the date in weblogs written in different languages. An improvement would be to identify the language of the document (e.g. with Apache Tika) and style the date following the locale of the identified language. Concerning future work, the approach can be altered and deployed in a supervised manner as well. In that case, the manual labelling of HTML elements will allow running the information extraction model on websites without the requirement for web feeds. Finally, another idea worth considering is to keep feeds for labelling data and to develop more robust ways of generating XPath expressions. We intend to explore the above opportunities in the future.

5 Conclusions

In this paper, we have presented a method for fully automated weblog wrapper generation. The generated wrapper exhibits increased granularity, since it manages to identify and extract several weblog properties, such as the title, author, publication date and main content of the post. This is accomplished through the induction of rules, which are selected following a probabilistic approach based on their scoring. Devising these rules is based on the generation of filters. The filters constitute a structure that, when applied to a web document, singles out an HTML element. They are described in tuples, where each of its element-attributes describes the HTML element in different forms (Absolute Path, CSS Classes and HTML IDs). The overall approach is evaluated against a real-world collection of weblogs and the results show that the wrappers generated are robust and efficient in handling different types of weblogs.

Acknowledgments. This work was conducted as part of the BlogForever project funded by the European Commission Framework Programme 7 (FP7), grant agreement No.269963.

References

1. Adelberg, B.: NoDoSE–a tool for semi-automatically extracting structured and semistructured data from text documents. SIGMOD Rec. 27(2), 283–294 (1998)
2. Baumgartner, R., Flesca, S., Gottlob, G.: Visual Web Information Extraction with Lixto. In: Proceedings of the 27th International Conference on Very Large Data Bases, pp. 119–128. Morgan Kaufmann Publishers, San Francisco (2001)
3. Baumgartner, R., Gatterbauer, W., Gottlob, G.: Web data extraction system. In: Encyclopedia of Database Systems, pp. 3465–3471. Springer (2009)
4. Crescenzi, V., Mecca, G., Merialdo, P.: Roadrunner: Towards automatic data extraction from large web sites. In: Proceedings of the International Conference on Very Large Data Bases, pp. 109–118 (2001)
5. Dutton, W., Blank, G.: Next generation users: The internet in Britain (2011)
6. Finkel, J., Grenager, T., Manning, C.: Incorporating non-local information into information extraction systems by gibbs sampling. In: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, pp. 363–370. Association for Computational Linguistics (2005)
7. Geibel, P., Pustylnikov, O., Mehler, A., Gust, H., Kühnberger, K.-U.: Classification of documents based on the structure of their DOM trees. In: Ishikawa, M., Doya, K., Miyamoto, H., Yamakawa, T. (eds.) ICONIP 2007, Part II. LNCS, vol. 4985, pp. 779–788. Springer, Heidelberg (2008)
8. Kohlschütter, C., Fankhauser, P., Nejdl, W.: Boilerplate detection using shallow text features. In: Proceedings of the Third ACM International Conference on Web Search and Data Mining, WSDM 2010, pp. 441–450. ACM, New York (2010)
9. Kushmerick, N.: Wrapper induction: Efficiency and expressiveness. Artificial Intelligence 118(1), 15–68 (2000)
10. Laender, A., Ribeiro-Neto, B., Da Silva, A., Teixeira, J.: A brief survey of web data extraction tools. ACM Sigmod Record 31(2), 84–93 (2002)
11. Liu, L., Pu, C., Han, W.: XWrap: An extensible wrapper construction system for internet information. In: Proceedings of the 16th International Conference on Data Engineering (ICDE 2000), San Diego, CA, pp. 611–621. IEEE (2000)

12. Muslea, I., Minton, S., Knoblock, C.: Hierarchical wrapper induction for semistructured information sources. Autonomous Agents and Multi-Agent Systems 4(1), 93–114 (2001)
13. Oita, M., Senellart, P.: Archiving data objects using Web feeds. In: Proceedings of International Web Archiving Workshop, Vienna, Austria, pp. 31–41 (2010)
14. Pennock, M., Davis, R.: ArchivePress: A Really Simple Solution to Archiving Blog Content. In: Sixth International Conference on Preservation of Digital Objects (iPRES 2009), California Digital Library, San Francisco, USA (October 2009)
15. Winkler, W.E.: String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. In: Proceedings of the Section on Survey Research Methods American Statistical Association, pp. 354–359 (1990)
16. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann (2005)
17. Yates, A., Cafarella, M., Banko, M., Etzioni, O., Broadhead, M., Soderland, S.: Textrunner: Open information extraction on the web. In: Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp. 25–26 (2007)
18. Zhai, Y., Liu, B.: Web data extraction based on partial tree alignment. In: Proceedings of the 14th international conference on World Wide Web, pp. 76–85. ACM (2005)

Author Index