COMAD 2012

Proceedings of the 18th International Conference on Management of Data

December 14-16, 2012
Pune, India

Editors

Amr El Abbadi
University of California, Santa Barbara, USA

Karin Murthy
IBM Research, India

Arnab Bhattacharya
Indian Institute of Technology, Kanpur, India

COMAD 2012

Sponsors and Facilities

Gold Sponsors

SAP

YAHOO! INDIA RESEARCH & DEVELOPMENT

Silver Sponsor

IBM

Bronze Sponsor

AEROSPIKE

Facilities

PERSISTENT

Infosys®

PREFACE

For close to two decades, the International Conference on Management of Data (COMAD), modeled along the lines of ACM SIGMOD, has been the premier international database conference hosted in India. The first COMAD was held in Hyderabad in 1989, and the most recent version was hosted in Bangalore in December 2011. The 18th edition in the COMAD series is held at the campus of Persistent Systems in Pune, Maharashtra from December 14-16, 2012. Named Queen of the Deccan, Oxford of the East, and cultural capital of Maharashtra, Pune is known for its educational facilities.

COMAD seeks to provide the community of researchers, practitioners, developers and users of data management technologies, a forum to present and discuss problems, solutions, innovations, experiences and emerging trends. During the past few years, the scope of COMAD has expanded to include, in addition to traditional database areas, topics in web technologies, information retrieval, and data mining.

This year's call for papers attracted 29 research submissions from across the world. Each research paper was rigorously reviewed by three members of the program committee, which featured 26 data management experts from academia and industry from 8 different countries. After in-depth discussions, we selected 7 high-quality papers for presentation at the conference. The authors of accepted research papers come from France, India, Japan, Switzerland, and the US, show-casing COMAD's international appeal. In addition to regular research papers we also accepted 5 work-in-progress presentations to give young researchers a chance to receive feedback from experts in their field.

COMAD 2012 features three keynote talks by Dr. Rakesh Agrawal (Head of Search Labs, Microsoft Research, USA), Sihem Amer-Yahia (Principal Research Scientist, CNRS, Laboratoire d'Informatique de Grenoble, France), and Rajeev Rastogi (Director of Machine Learning, Amazon, India). The keynotes span topics from advanced technology for education and social media analytics to web-scale information extraction. The program also hosts 4 tutorials from leading international experts covering big data technologies, Markov logic networks, spatio-temporal indexing, and data fusion. Finally, to round out the academic program we invited 10 posters and demos from the academic community in India. We also continued the tradition started by COMAD in 2010 to invite Indian authors of papers published in premier international conferences to present their work at COMAD. This year features three papers from VLDB 2012 and one paper from SIGMOD 2012.

In addition to academic talks, presentations, and tutorials COMAD 2012 also features two invited industry talks and two sponsor talks, giving participants a chance to interact with leading industry experts. To ensure visibility of COMAD beyond this conference, these proceedings will also be available through ACM SIGMOD and DBLP.

We would like to thank all the members of the COMAD Organizing Committee and the COMAD Program Committee for their generous support, enabling us to put together such a high-quality program. We are also grateful for the support and generosity of our sponsors. Without our gold sponsors SAP and Yahoo as well as our silver sponsor IBM this conference would not be possible. We also thank Persistent Systems for providing a campus for the conference and Infosys for providing free accommodation at their guest house. Finally, we acknowledge the sustained cooperation and assistance

extended by the Computer Society of India in organizing this event.

In closing, we welcome you to the COMAD 2012 conference in Pune and hope you will have a fruitful and stimulating experience.

Chandrashekhar Sahasrabudhe
Persistent Systems, Pune, India
(General Chair)

Amr El Abbadi
University of California, Santa Barbara, USA
Karin Murthy
IBM Research - India
(Program Co-Chairs)

Arnab Bhattacharya
Indian Institute of Technology, Kanpur, India
(Proceedings Chair)

ORGANIZING COMMITTEE

General Chair
- Chandrashekhar Sahasrabudhe, *Persistent Systems, Pune*

Program Chairs
- Amr El Abbadi, *University of California, Santa Barbara*
- Karin Murthy, *IBM Research - Bangalore*

CSI Div II Chair
- T. V. Gopal, *Anna University, Chennai*

Work in Progress Chair
- Srikanta Bedathur, *IIIT Delhi*

Industry Chair
- Sudarshan Murthy, *The Else Institute, Portland, Oregon*

Tutorial Chair
- Maya Ramanath, *IIIT Delhi*

Demo Chair
- Sriram Raghavan, *IBM Research - Bangalore*

Panel Chair
- Sharma Chakravarthy, *The University of Texas at Arlington*

Publication Chair
- Arnab Bhattacharya, *IIT Kanpur*

Organizing Committee Chair
- Arun Kadekodi, *Soft Corner, Pune*

Web Chair
- Anand Joglekar, *Ameya Software, Pune*

Host Chapter Chair
- Amit Danglé, *National School of Leadership, Pune*

PROGRAM COMMITTEE

- Alfredo Cuzzocrea, *Italian National Research Council, Italy*
- Anirban Mondal, *IIIT Delhi, New Delhi, India*
- Arvind Arasu, *Microsoft Research, Seattle, Washington*
- Balaraman Ravindran, *IIT Madras, Chennai, India*
- Chetan Gupta, *HP Labs, Palo Alto, California*
- Fan Wang, *Microsoft Corporation, Bellevue, Washington*
- Hoda Mokhtar, *Cairo University, Egypt*
- Maitreya Natu, *Tata Consultancy Services, Pune, India*
- Mohamed Mokbel, *University of Minnesota, Minneapolis*
- P. Sreenivasa Kumar, *IIT Madras, Chennai, India*
- Rajiv Ranjan, *The University of New South Wales, Sydney, Australia*
- Ralf Schenkel, *Max Planck Institute, Saarbruecken, Germany*
- Ramanujam Halasipuram, *IBM Research – Bangalore, India*
- Sameep Mehta, *IBM Research – Delhi, India*
- Sanjay Chawla, *The University of Sydney, Australia*
- Shridhara Aithal, *The Else Institute, Bangalore, India*
- Sharma Chakravarthy, *The University of Texas at Arlington, Texas*
- Sitaram Asur, *HP Labs, Palo Alto, California*
- Sourav S. Bhowmick, *Nanyang Technological University, Singapore*
- Sudipto Das, *Microsoft Research, USA*
- Sumit Negi, *IBM Research – Delhi, India*
- Sunil Prabhakar, *Purdue University, West Lafayette, Indiana*
- Qiong Luo, *Hong Kong University of Science and Technology, Hong Kong*
- Vaishali P. Sadaphal, *Tata Consultancy Services, Pune, India*
- Vivek Narasayya, *Microsoft Research, Seattle, Washington*
- Walid Aref, *Purdue University, Indiana*

CONTENTS

Work in Progress

K E Y N O T E S

Reimagining Textbooks Through the Data Lens

Rakesh Agrawal
Search Labs, Microsoft Research

Abstract

Textbooks are the primary vehicles for delivering subject knowledge to the students and are known to be the educational input most consistently associated with improvements in student learning. With the emergence of abundant online content, cloud computing, and electronic reading devices, textbooks are poised for transformative changes. Inspired by the emergence of the electronic medium for “printing” and “distributing” textbooks, we present our early explorations into developing a data mining based approach for enhancing the quality of electronic textbooks. Specifically, we first describe a diagnostic tool for authors and educators to algorithmically identify deficiencies in textbooks. We then discuss techniques for algorithmically augmenting different sections of a book with links to selective content mined from the Web.

Our tool for diagnosing deficiencies consists of two components. Abstracting from the education literature, we identify the following properties of good textbooks: (1) *Focus*: Each section explains few concepts, (2) *Unity*: For every concept, there is a unique section that best explains the concept, and (3) *Sequentiality*: Concepts are discussed in a sequential fashion so that a concept is explained prior to occurrences of this concept or any related concept. Further, the tie for precedence in presentation between two mutually related concepts is broken in favor of the more significant of the two. The first component provides an assessment of the extent to which these properties are followed in a textbook and quantifies the comprehension load that a textbook imposes on the reader due to non-sequential presentation of concepts [1]. The second component identifies sections that are not written well and can benefit from further exposition. We propose a probabilistic decision model for this purpose, which is based on the syntactic complexity of writing and the notion of the dispersion of key concepts mentioned in the section [3].

For augmenting a section of a textbook, we first identify the set of key concept phrases contained in a section. Using these phrases, we find web articles that represent the central concepts presented in the section and endow the section with links to them [4]. We also describe techniques for finding images that are most relevant to a section of the textbook, while respecting the constraint that the same image is not repeated in different sections of the same chapter. We pose this problem of matching images to sections in a textbook chapter as an optimization problem and present an efficient algorithm for solving it [2].

We finally provide the results of applying the proposed techniques to a corpus of widely-used, high school textbooks published by the National Council of Educational Research and Training, India. We consider books from grades IX--XII, covering four broad subject areas, namely, Sciences, Social Sciences, Commerce, and Mathematics. The preliminary results are encouraging and indicate that developing technological approaches to embellishing textbooks could be a promising direction for research.

References

[1] R. Agrawal, S. Chakraborty, S. Gollapudi, A. Kannan, and K. Kenthapadi. Empowering authors to diagnose comprehension burden in textbooks. In KDD, 2012.
[2] R. Agrawal, S. Gollapudi, A. Kannan, and K. Kenthapadi. Enriching textbooks with images. In CIKM, 2011.
[3] R. Agrawal, S. Gollapudi, A. Kannan, and K. Kenthapadi. Identifying enrichment candidates in textbooks. In WWW, 2011.
[4] R. Agrawal, S. Gollapudi, K. Kenthapadi, N. Srivastava, and R. Velu. Enriching textbooks through data mining. In ACM DEV, 2010.

Biography

Dr. Rakesh Agrawal is a Microsoft Technical Fellow, heading the Search Labs in Microsoft Research in Silicon Valley. He is a Member of the National Academy of Engineering, a Fellow of ACM, and a Fellow of IEEE. He is the recipient of the ACM-SIGKDD First Innovation Award, ACM-SIGMOD Innovations Award, IIT-Roorkee Distinguished Alumni Award, ACM-SIGMOD Test of Time Award, VLDB 10-Yr Most Influential Paper Award, and ICDE Most Influential Paper Award. Scientific American named him to the list of 50 top scientists and technologists in 2003. Dr. Agrawal has been granted more than 60 patents and has published more than 150 research papers. He has written the first and second highest cited papers in the fields of databases and data mining. Before Microsoft, he worked as an IBM Fellow at IBM Almaden and at Bell Laboratories, Murray Hill. He received his Ph.D. degree in Computer Science from the University of Wisconsin-Madison.

The 18th International Conference on Management of Data (COMAD),
14th-16th Dec, 2012 at Pune, India.

User Activity Analytics on the Social Web of News

Sihem Amer-Yahia
Qatar Computing Research Center

Abstract

The proliferation of social media is undoubtedly changing the way people produce and consume news online. Editors and publishers in newsrooms need to understand user engagement and audience sentiment evolution on various news topics. News consumers want to explore public reaction on articles relevant to a topic and refine their exploration via related entities, topics, articles and tweets. I will present MAQSA, a system for social analytics on news. MAQSA provides an interactive topic-centric dashboard that summarizes social activity around news articles. The dashboard contains an annotated comment timeline, a social graph of comments, and maps of comment sentiment and topics. The analysis of both content and user engagement in social media in MAQSA enables the exploration of new ways of immersing users in a news consumption experience.

Biography

Sihem Amer-Yahia is Principal Research Scientist at Qatar Computing Research Center (QCRI) and DR1 CNRS at LIG in Grenoble. Sihem's interests are at the intersection of large-scale data management and analytics, and social content at large. Until May 2011, she was Senior Scientist at Yahoo! Research for 5 years and worked on revisiting relevance models and top-k processing algorithms on datasets from Delicious, Yahoo! Personals and Flickr. Before that, she spent 7 years at AT&T Labs in NJ, working on XML query optimization and XML full-text search. Sihem is editor of the W3C XML full-text standard. She is a member of the VLDB Endowment and the ACM SIGMOD executive committee. Sihem is track chair at PVLDB and SIGIR this year. She serves on the editorial boards of ACM TODS, the VLDB Journal and the Information Systems Journal. Sihem received her Ph.D. in Computer Science from Univ. Paris-Orsay and INRIA in 1999, and her Diplome d'Ingenieur from INI, Algeria in 1994.

The 18th International Conference on Management of Data (COMAD),
14th-16th Dec, 2012 at Pune, India.

Building Knowledge Bases from the Web

Rajeev Rastogi
Machine Learning, Amazon

Abstract

The web is a vast repository of human knowledge. Extracting structured data from web pages can enable applications like comparison shopping, and lead to improved ranking and rendering of search results. In this talk, I will describe two efforts to extract records from pages at web scale. The first is a wrapper induction system that handles end-to-end extraction tasks from clustering web pages to learning XPath extraction rules to relearning rules when sites change. The system has been deployed in production within Yahoo! to extract more than 500 million records from ~200 web sites. The second effort exploits machine learning models to automatically extract records without human supervision. Specifically, we use Markov Logic Networks (MLNs) to capture content and structural features in a single unified framework, and devise a fast graph-based approach for MLN inference.

Biography

Rajeev Rastogi is the Director of Machine Learning at Amazon. Previously, he was the Vice President of Yahoo! Labs Bangalore, and a Bell Labs Fellow at Bell Labs in Murray Hill, NJ. Rajeev is active in the fields of databases, data mining, and networking, and has served on the program committees of several conferences in these areas. He currently serves on the editorial board of the CACM, and has been an Associate editor for IEEE Transactions on Knowledge and Data Engineering in the past. He has published over 125 papers, and holds over 50 patents. Rajeev received his B. Tech degree from IIT Bombay, and a PhD degree in Computer Science from the University of Texas, Austin.

The 18th International Conference on Management of Data (COMAD),
14th-16th Dec, 2012 at Pune, India.

T U T O R I A L S

Spatio-Temporal Indexing - Current Scenario, Challenges and Approaches

Aditya Telang, Deepak Padmanabhan, Prasad Deshpande
IBM Research – India
Bangalore, INDIA
{adtelang, deepak.s.p, prasdesh}@in.ibm.com

1. MOTIVATION

With rapid advancements in computing hardware, tracking devices such as GPS receivers and sensors have become pervasive, generating a large amount of spatio-temporal data, such as measurements of temperature, pressure, air quality, traffic, etc. using sensors, GPS data from mobile phones and data from radars that capture location information about people and other moving objects such as cars and aeroplanes. This has enabled a wide variety of spatio-temporal applications, resulting in a renewed interest in techniques for handling spatio-temporal data. Over the past two decades or so, a large number of indexes for supporting spatial, temporal and spatio-temporal data have been independently proposed in the database and data mining communities. However, there exists no clear-cut guidelines or a prescriptive formula for pointing out which index should be chosen when specific needs of the underlying application are known. In addition, since spatio-temporal indexes have been proposed under various domains, it is hard for researchers and practitioners to determine whether some specified indexes are indeed available to address the problem at hand. For instance, an index like PO-Tree [7] is suitable for monitoring static spatio-temporal objects (such as sensors, cell-phone towers, etc.) but it is completely undesirable for handling moving object data (e.g., location tracking of cell-phone users, GPS tracking of vehicles and so on). Likewise, if the semantics of the application require indexing trajectories of moving objects, only a specific set of indexes (such as PA-Tree [6]) are useful whereas others such as (APR-Tree [3]) are undesirable.

We design this tutorial to expose the audience to the vast reservoir of spatio-temporal indexing techniques that are available in literature. In addition, apart from introducing the various indexes, our aim is to analyze the pros and cons of different indexing mechanisms when applied to various diverse scenarios. Given the large recent interest in spatio-temporal data analytics among corporates and academia, we hope that this tutorial is well-positioned in time to enhance and enrich the understanding of spatio-temporal data processing. Further, we think that the subject matter of this tutorial is a perfect fit for the COMAD conference that has a focused track for data management and its disciplines.

The 18th International Conference on Management of Data (COMAD), 14th-16th Dec, 2012 at Pune, India.

2. TUTORIAL ORGRANIZATION

We propose to organize this tutorial for a duration of 3 hours. A brief outline of the organization of the tutorial is as follows:

1. **Motivation**: (10 minutes)
2. **Spatial Indexing**: (30 minutes)
   - Spatial Data Types
   - Spatial Query Categories
   - Classification of Spatial Indexes
     - Grid-based technique
     - Tree-based technique
   - Analysis of Different Spatial Indexes (such as Geodesic Grid, R Tree, KD-Tree and so on)
     - Semantics of each index
     - Typical Usage
     - Applicability for real-time applications
       - Pros & Cons
3. **Temporal Indexing**: (20 minutes)
   - Need for temporal indexing
   - Types of Temporal Indexes
     - Semantics of each index
     - Typical Usage
     - Applicability for real-time applications
       - Pros & Cons
4. **Spatio-temporal Indexing**: (75 minutes)
   - Motivation & Basic Techniques
     - Native Space Indexing
     - Parametric Space Indexing
   - Types of Spatio-Temporal Indexes
     - Semantics of each index
     - Typical Usage
     - Applicability for real-time applications
       - Pros & Cons
5. **Comparative Analysis**: (35 minutes)
   - Different Spatio-Temporal Application Scenarios
     - Application of Different Indexes
     - Implications
     - Pros & Cons

6. **Conclusion & Discussion** (10 minutes)
    - Summary of the Tutorial
    - Pointers to Exciting New Problems

A set of transparencies(in PDF format) and a recommendation of papers will be made available to the participants.

3. TUTORIAL CONTENT

Here we present a detailed description of the material presented in this tutorial.

3.1 Motivation

In this segment, we introduce the problem of managing and handling spatio-temporal data. We present the different types of query scenarios that are typically posed on such data and illustrate the need for indexing mechanisms for organizing this data for effective retrieval of results. We provide a brief overview of the different contexts in which spatio-temporal data management has been addressed i.e., organizing historical data for analysis, warehousing data for mining, maintaining real-time data for frequent updates and queries, isolating and organizing trajectory data as well as individual data points for moving objects, and so on.

3.2 Spatial Indexing

In this part of the tutorial, we dig deeper into different types of spatial indexes (such as Geodesic Grid, R-tree [4], R+Tree [12], R*Tree [11], KD-Tree [8] along with its derivatives such as the Quad-Tree [9] and Oct-Tree [5]) that have been purely proposed for organizing different kinds of spatial data such as – surface of the earth (e.g., volcanic zones, earthquake regions, etc.), natural entities (e.g., forests, rivers, etc.), man-made entities (e.g., universities, castles, etc.) and moving spatial entities (e.g., cars on roads, ships in oceans, etc.). We discuss the semantics associated with each index . Specifically, we demonstrate how each of this index behaves when subjected to standard paradigms of spatial querying i.e., range queries and k-nearest neighbour queries. Further, we also provide insights as to which index to select (i.e., either a grid-based or a tree-based) depending on the needs of the problem setting.

3.3 Temporal Indexing

Similar to spatial indexing, temporal indexing has received a lot of attention for organizing database tuples based on their timestamps. We briefly touch base with Allen's Algebra [2] in order to understand the type of temporal queries typically issued on databases. Accordingly, we survey the different temporal indexing techniques and their performance aspects when handling such queries.

3.4 Spatio-Temporal Indexing & Comparative Analysis

This section forms the core component of this tutorial. We elaborate of the different types of indexing techniques for different kinds of spatio-temporal needs i.e., indexing data for statistical analysis, organizing trajectory-related data, managing data with respect to constantly moving and frequently updating objects, and so on. We discuss the semantics of each of these techniques, and provide a comparative analysis of different spatio-temporal indexing mechanisms (such as the TPR-Tree [10], the TPR*-Tree [13], the COLR-Tree [1], the Q+R-Tree [14] and others such as RT-Tree, 3DR-Tree, MV3R-Tree, HR-Tree, etc [3]) with respect to their performance, their ability to support range and kNN queries, and their overall applicability to different kinds of real-time monitoring of moving objects in the context of a spatio-temporal setting.

3.5 Analysis and Conclusions

Here, we summarize the contents of the tutorial and present a various pointers for future work.

4. TARGETED AUDIENCE & EXPECTATIONS

This tutorial is mainly targeted at several kinds of audience such as researchers, graduate students and industry professionals working in and/or interested in the area of handling, maintaining and working with spatio-temporal data in the context of real-time applications. The tutorial is organized in a self-contained way and does not assume any particular expertise from the audience. At the end of the tutorial, we hope that the attendees will be equipped with insights into different aspects involved in indexing spatio-temporal data, and would have a clear picture in terms of what indexing techniques to select for specific needs of applications using such data. We attempt our best to maintain a striking balance between theoretical concepts and practical importance of the problems in the tutorial. Thus, we hope that practitioners also get benefited from this tutorial.

5. BRIEF BIOGRAPHY

Aditya Telang: Aditya is a researcher at IBM Research India since 2011. Prior to joining IBM, he finished his PhD from University of Texas at Arlington. His current research interests include Spatio-Temporal Data Analytics, Information Management, and Business Analytics.

Deepak Padmanabhan: Deepak works with the Information Management Group at IBM Research India at Bangalore. He obtained his masters degree from IIT Madras prior to joining IBM.

Prasad Deshpande: Prasad M Deshpande is a Senior Researcher at IBM Research - India and Manager of the Information Analytics group. His areas of expertise lie in data management, specifically data integration and warehousing, OLAP, data mining and text analytics. He received a B. Tech in Computer Science and Engineering from IIT, Bombay and a M.S. and Ph.D. in Database systems from the University of Wisconsin, Madison. He has worked at several companies, including startups, IBM Almaden Research Center and currently at IBM Research - India. He has more than 35 publications in reputed conferences and journals and has several patents to his name. He has served on the Program Committee of many conferences, most recently being the PC Chair for COMAD 2011 and ACM Compute 2010.

6. REFERENCES

[1] Y. Ahmad and S. Nath. Colr-tree: Communication-efficient spatio-temporal indexing for a sensor data web portal. In *ICDE*, pages 784–793, 2008.

[2] J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[3] H.-J. Cho, J.-K. Min, and C.-W. Chung. An adaptive indexing technique using spatio-temporal query workloads. *Information & Software Technology*, 46(4):229–241, 2004.

[4] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984.

[5] C. L. Jackins and S. L. Tanimoto. Quad-trees, oct-trees, and k-trees: A generalized approach to recursive decomposition

of euclidean space. *IEEE Trans. Pattern Anal. Mach. Intell.*, 5(5):533–539, 1983.

[6] J. Ni and C. V. Ravishankar. Pa-tree: A parametric indexing scheme for spatio-temporal trajectories. In *SSTD*, pages 254–272, 2005.

[7] G. Nol, S. Servigne, and R. Laurini. The po-tree: a real-time spatiotemporal data indexing structure. In *11th International Symposium on Spatial Data Handling*, pages 259–270, 2005.

[8] B. C. Ooi. Spatial kd-tree: A data structure for geographic database. In *BTW*, pages 247–258, 1987.

[9] M. H. Overmars and J. van Leeuwen. Dynamic multi-dimensional data structures based on quad- and *k - d* trees. *Acta Inf.*, 17:267–285, 1982.

[10] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD Conference*, pages 331–342, 2000.

[11] T. K. Sellis. Review - the r*-tree: An efficient and robust access method for points and rectangles. *ACM SIGMOD Digital Review*, 2, 2000.

[12] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. In *VLDB*, pages 507–518, 1987.

[13] Y. Tao, D. Papadias, and J. Sun. The tpr*-tree: an optimized spatio-temporal access method for predictive queries. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pages 790–801. VLDB Endowment, 2003.

[14] Y. Xia and S. Prabhakar. Q+rtree: Efficient indexing for moving object database. In *DASFAA*, pages 175–182, 2003.

Big Data Technologies circa 2012

Vinayak Borkar*
University of California, Irvine
vborkar@ics.uci.edu

Michael J. Carey†
University of California, Irvine
mjcarey@ics.uci.edu

1. INTRODUCTION

The growth of the World Wide Web has led to an astronomical amount of data being generated. More recently, the amount of user-generated content has seen tremendous expansion thanks to social media like Facebook and Twitter. Enterprises, researchers, and even governments consider this data to be an invaluable source of insight into people's behavior, creating a race to analyze as much data as possible. This race has driven virtually everyone, ranging from Web companies to brick and mortar businesses, into a "Big Data" frenzy. On the systems side, traditional relational databases have proven to be un-scalable, too expensive, too rigid, and/or too heavy-weight for dealing with current Big Data problems. As a result, there has been an explosion in the number of systems being developed, both within industry as well as in academia, to manage massive amounts of data.

Traditionally, data management systems were classified broadly into Online Transaction Processing (OLTP) systems and Decision Support Systems (DSS). Key-Value stores [13] have become the system of choice in the Big Data universe to perform short, single-record "transactions" at scale, playing the role of OLTP systems, albeit with limited functionality and weaker transaction guarantees. On the analytics side, MapReduce [17] and Hadoop [5] have dominated the space for scalable data analyses. There has also been an emergence of specialized systems for Big Data problems that are not naturally solved by MapReduce (those involving iterations, for example).

2. BIG DATA BACKGROUND

Google, being at the forefront of the Big Data "revolution", was forced to take matters into its own hands to stay competitive in the search engine space. Falling costs of commodity hardware made it evident that the only way to reign in the growing data was to use many computers in parallel. In 2004, Google proposed the MapReduce [17] system in conjunction with the Google File System [22] as a way to perform computation at massive scale using commodity computers. The MapReduce framework greatly simplified parallel computation for programmers by letting them avoid parallel programming. Programmers simply had to implement simple single-threaded code in the form of "Map" and "Reduce" functions which was invoked by the MapReduce infrastructure in parallel on different instances of data spread across Google's distributed file system. In addition to being able to index the entire web in reasonable amounts of time, the MapReduce system allowed programmers at Google to perform massive data processing tasks quickly using a simple programming model. Yahoo!, motivated by the MapReduce system from Google, implemented Hadoop [5] (and the Hadoop Distributed File system) and released it as open-source software.

The MapReduce paper marked the beginning of a new era of Big Data technologies. High-level layers were soon developed on top of MapReduce, further increasing programmer productivity for domain-specific tasks. Sawzall [29] and (much later) Tenzing [25] were two systems built by Google using the MapReduce layer as a runtime and parallelizing framework for text-processing and SQL execution, respectively. Outside of Google, Hadoop has become the de-facto standard for scaling data-processing and Yahoo! created the PigLatin [27] language and the Pig [28] system to run on top of Hadoop. Facebook released Apache Hive [6], a SQL-like language that also uses Hadoop as the runtime layer. Besides the Hadoop/MapReduce family of systems, alternate large-scale data-processing frameworks were proposed by various companies as well as research groups at universities. Some examples of alternative technologies include Dryad [24], DryadLINQ [32], and SCOPE [14] from Microsoft, Nephele/PACTs [7] from TU Berlin, Hyracks [10] and ASTERIX [8] from the UC Irvine, and Spark [33] from UC Berkeley.

While the MapReduce approach has been successful at analyzing large datasets that are rarely modified, there was also a need for systems to store large amounts of data and perform quick inserts, updates, and deletes of records identified by a key. This requirement led to the introduction of Key-Value stores into the Big Data ecosystem. Google developed BigTable [15], Amazon created Dynamo [18], Facebook created the Dynamo clone, Cassandra [1], and Yahoo! created the BigTable clone, HBase [2] as well as a new system, PNUTS [16] to satisfy this growing need.

*Presenter
†Co-author of tutorial content, but not presenting at the conference

The 18th International Conference on Management of Data (COMAD), 14th-16th Dec, 2012 at Pune, India.

Today's Big Data systems also include specialized platforms for solving niche problems. Pregel [26] and its open-source clones (Giraph [4] and GoldenOrb [23]) are used for parallel computation over large graphs. Similar in spirit to the MapReduce programming model, Pregel provides a simple API for programmers to express complicated graph algorithms using single-threaded code (the logic for a single vertex) which is then parallelized by the Pregel infrastructure. Machine-Learning has been another domain that has seen the emergence of specialized systems based on the Iterative-Map-Reduce-Update model [12]. Vowpal Wabbit [3] is a system custom built at Yahoo! for solving Machine Learning problems involving aggregation trees.

No list of Big Data Technologies can be considered complete without the mention of Parallel Databases, a heavily researched [20, 9, 21] area in the period from the early 1980s to the mid 1990s. Commercially, Teradata [30] and NonStop SQL [31] were tremendous successes in the parallel database space. DeWitt and Gray [19] describe important principles surrounding partitioned-parallel data computation using shared-nothing computers; the very same principles govern the operation of all the "modern" Big Data systems mentioned earlier in this section. A longer discussion of Big Data technologies can be found in [11].

3. TUTORIAL OUTLINE

The outline for the tutorial is as follows:

1. Background: Parallel Database Systems
   - Shared Everything vs. Shared Disk vs. Shared Nothing Systems
   - Three Forms of Parallelism in Parallel Database Systems: Pipelined Parallelism, Partitioned Parallelism, and Independent Parallelism
   - Parallelization Metrics: Speedup and Scaleup
   - A Case Study: Gamma
2. MapReduce and Hadoop
   - The MapReduce Programming Model
   - The Hadoop Platform
     - Hadoop Distributed File System (HDFS)
     - Fault-Tolerance in MapReduce
   - Examples
     - Word Count
     - Join and Aggregate Processing
3. High-Level Languages for Big Data
   - PigLatin
   - HiveQL
   - ASTERIX Query Language (AQL)
4. Alternative Data-Parallel Platforms
   - Overview of the Space of Big Data Platforms
   - Case Studies
     - Hyracks
     - Stratosphere (Nephele/PACTs)
5. Key-Value Stores
   - Key Value API
   - Consistency in Key-Value Stores
   - Case Studies
     - Cassandra
     - HBase
     - PNUTS
6. Specialized Systems
   - Pregel
   - Iterative-Map-Reduce-Update

4. PRESENTER BIO

Vinayak Borkar is a PhD. candidate and a Research Scientist at the University of California, Irvine in the Computer Science department. His research focuses on the efficient use of large clusters in solving Big Data problems. He was the primary developer of the Hyracks data-parallel platform. Prior to his affiliation with UCI, he developed various data-management products for close to ten years at Informatica Inc., BEA Systems Inc., and several startups. He received his Masters in Computer Science and Engineering from the Indian Institute of Technology, Bombay in 2001.

5. REFERENCES

[1] Apache Cassandra website. `http://cassandra.apache.org`.

[2] Apache HBase website. `http://hbase.apache.org`.

[3] Vowpal wabbit. `http://hunch.net/ vw/`.

[4] Giraph: Open-source implementation of Pregel. http://incubator.apache.org/giraph/.

[5] Hadoop: Open-source implementation of MapReduce. http://hadoop.apache.org.

[6] The Hive Project. http://hive.apache.org/.

[7] Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl, and Daniel Warneke. Nephele/PACTs: a Programming Model and Execution Framework for Web-Scale Analytical Processing. In *SoCC*, pages 119–130, New York, NY, USA, 2010. ACM.

[8] Alexander Behm, Vinayak R. Borkar, Michael J. Carey, Raman Grover, Chen Li, Nicola Onose, Rares Vernica, Alin Deutsch, Yannis Papakonstantinou, and Vassilis J. Tsotras. Asterix: towards a scalable, semistructured data platform for evolving-world models. *Distrib. Parallel Databases*, 29:185–216, June 2011.

[9] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez. Prototyping Bubba, A Highly Parallel Database System. *IEEE Trans. on Knowl. and Data Eng.*, 2(1):4–24, March 1990.

[10] Vinayak R. Borkar, Michael J. Carey, Raman Grover, Nicola Onose, and Rares Vernica. Hyracks: A Flexible and Extensible Foundation for Data-Intensive Computing. In *ICDE*, pages 1151–1162, 2011.

[11] Vinayak R. Borkar, Michael J. Carey, and Chen Li. Inside "Big Data Management": Ogres, Onions, or Parfaits? In *EDBT*, 2012.

[12] Yingyi Bu, Vinayak Borkar, Michael J. Carey, Joshua Rosen, Neoklis Polyzotis, Tyson Condie, Markus Weimer, and Raghu Ramakrishnan. Scaling datalog for machine learning on big data. Technical report, CoRR. URL: http://arxiv.org/submit/427482 or http://isg.ics.uci.edu/techreport/TR2012-03.pdf, 2012.

[13] Rick Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Rec.*, 39:12–27, May 2011.

[14] Ronnie Chaiken, Bob Jenkins, Per A. Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets. *Proc. VLDB Endow.*, 1(2):1265–1276, 2008.

[15] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, June 2008.

[16] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1(2):1277–1288, August 2008.

[17] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI '04*, pages 137–150, December 2004.

[18] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, October 2007.

[19] David DeWitt and Jim Gray. Parallel Database Systems: The Future of High Performance Database Systems. *Commun. ACM*, 35:85–98, June 1992.

[20] David J. DeWitt, Robert H. Gerber, Goetz Graefe, Michael L. Heytens, Krishna B. Kumar, and M. Muralikrishna. GAMMA - a high performance dataflow database machine. In *VLDB*, pages 228–237, 1986.

[21] Shinya Fushimi, Masaru Kitsuregawa, and Hidehiko Tanaka. An Overview of The System Software of a Parallel Relational Database Machine GRACE. In *Proceedings of the 12th International Conference on Very Large Data Bases*, VLDB '86, pages 209–219, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc.

[22] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *Proc. 19th ACM Symp. on Operating Systems Principles*, SOSP '03, New York, NY, USA, 2003. ACM.

[23] GoldenOrb: Open-source implementation of Pregel. http://www.raveldata.com/goldenorb/.

[24] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *EuroSys*, pages 59–72, 2007.

[25] Liang Lin, Vera Lychagina, and Michael Wong. Tenzing A SQL Implementation on the MapReduce Framework. *Proceedings of the VLDB Endowment*, 4(12):1318–1327, 2011.

[26] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.

[27] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig Latin: A Not-so-Foreign Language for Data Processing. In *SIGMOD Conference*, pages 1099–1110, 2008.

[28] Pig Website. `http://hadoop.apache.org/pig`.

[29] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the Data: Parallel Analysis with Sawzall. *Scientific Programming*, 13(4):277–298, 2005.

[30] J. Shemer and P. Neches. The Genesis of a Database Computer. *Computer*, 17(11):42 –56, Nov. 1984.

[31] The Tandem Database Group. Nonstop SQL: A distributed, high-performance, high-availability implementation of SQL. *Second International Workshop on High Performance Transaction Systems*, September 1987.

[32] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. In Richard Draves and Robbert van Renesse, editors, *OSDI*, pages 1–14. USENIX Association, 2008.

[33] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. HotCloud'10, page 10, Berkeley, CA, USA, 2010.

Markov Logic Networks: Theory, Algorithms and Applications

Parag Singla
Indian Institute of Technology Delhi
Hauz Khas, New Delhi
parags@cse.iitd.ac.in

ABSTRACT

Most real world problems are characterized by relational structure i.e. entities and relationships between them. Further, they are inherently uncertain in nature. Theory of logic gives the framework to represent relations. Statistics provides the tools to handle uncertainty. Combining the power of two becomes important for accurate modeling of many real world domains. Last decade has seen the emergence of a new research area popularly known as Statistical Relational Learning (SRL) which aims at achieving this merger. Markov logic is one of the most well-known SRL models which combines the power of first-order logic with Markov networks. The underlying domain is represented as a set of weighted first-order logic formulas. The associated weight of a formula represents the strength of the corresponding constraint. Higher the weight, stronger the constraint is. Markov logic theory can be seen as defining a template for constructing ground Markov networks, and hence, the name Markov logic networks.

Inference problem in Markov logic corresponds to finding the state of a subset of nodes (query) given the state of another subset of nodes (evidence) in the network. Learning corresponds to finding the optimal set of weights for the formulas as well as discovering the formulas themselves. Many of the standard algorithms for inference and learning in ground Markov networks do not scale well to the size of the networks that can be represented using Markov logic. Further, there is a rich template structure across ground formulas which can be exploited to devise efficient inference and learning algorithms. Due to their representational strength, availability of inference and learning algorithms, ease of use and the availability of an open source implementation, Markov logic has been effectively applied to a variety of application domains including entity resolution, web-mining, link prediction, social network analysis, image analysis, robotics, natural language processing and plan recognition, to cite a few.

This tutorial will cover in detail the theory behind Markov logic starting from the basics of first-order logic and Markov networks. We will also look at various inference and learning algorithms for Markov logic. Second half of the tutorial will focus on some of the applications to which Markov logic has been applied. We will look at the modeling aspect of the problem as well as actually writing up the theory using the open source software, Alchemy, which implements Markov logic framework.

Bio-Sketch

Parag is an undergraduate from IIT Bombay batch 2002. He studied at the University of Washington Seattle to get his Masters and PhD degrees. His PhD work focused on Markov logic, a formalism to combine the power of logic and probability. He has done some pioneering work in developing lifted inference techniques for Markov logic. He has also worked extensively in applying Markov logic to a variety of real world problems including entity resolution, link prediction, abductive plan recognition and vision related problems. His paper on a new technique for entity resolution using attribute-mediated dependences won the best paper award at PKDD 2005. After finishing his PhD in 2009, he spent a couple of years at UT Austin for a post-doc. He has been working as an Assistant Professor at IIT Delhi since December 2011. His current research work continues to focus on developing efficient inference and learning algorithms for SRL (statistical relational learning) models. He is also looking at their application to social network analysis and video activity recognition. Parag has over a dozen publications in top tier peer-reviewed international conferences and workshops, one best paper award and two patents to his name. He has been a reviewer for many reputed international journals and served on the program committee for several premiere international conferences including senior program committee for IJCAI-11 and program committees for AAAI-12 and ECAI-12.

The 18th International Conference on Management of Data (COMAD),
14th-16th Dec, 2012 at Pune, India.
Copyright ©2012 Computer Society of India (CSI).

Reliability Aware Data Fusion

Sameep Mehta
IBM Research India
New Delhi, India
sameepmehta@in.ibm.com

L. Venkata Subramaniam
IBM Research India
New Delhi, India
lvsubram@in.ibm.com

1. OVERVIEW

Due to ubiquitous sensors (GPS, Accelerometer), easy of use apps (Facebook, Twitter etc), presence of audio & video recording devices and higher internet connectivity, the key characteristics of raw data is changing. This new data can be characterized by 4Vs Volume, Velocity, Variety and Veracity. Moreover, due to popular trend of crowd sourcing or citizen sensors, it is reasonable to assume that people will provide multiple evidence of same event using different data types. For example during a Football match, some people will Tweet about Goals, Penalties etc while others will take a picture and upload it. Although the underlying modalities are different (text and image), the data describes the same event. Such multi modal evidences should be used to strengthen the belief in underlying physical event. Finally, each of the data point will have inherent uncertainty. The uncertainty can arise from inconsistent, incomplete, and ambiguous data as well as the trust worthiness of the user. Similarly, some sources are more reliable than others which will also play a part in overall reliability. The volume, velocity and variety are measurable/observable, however, there is no measure of truthfulness.

Traditionally, CS research has focused on Volume and Velocity. However, multimodal data fusion and reliability are less explored. Through this tutorial will wish to draw the attention of researchers towards these dimensions by presenting some real life use cases, highlighting the key technical challenges, existing techniques and need for new .

2. TOPICS

We intend to over the following topics during the tutorial

- Data Characteristics with 4V dimensions and use cases from Public Safety Domain.
- Key Technical Challenges (non exhaustive list)
  - Entity Resolution
  - Data Cleaning
  - Performance
  - Indexing and Storage
  - Updating of Data
  - Use Case: Generating Single View of Entity
- Data Fusion Methods
  - Probabilistic Data Fusion using Bayes Theorem,
  - Information Measures like Entropy, Mutual Information, Fisher Information
  - Interval Calculus, Fuzzy Logic and Evidential Reasoning
  - Kalman Filters & variants, Nearest Neighbor Filters and Probabilistic Data Association Filter
- Reliability
  - Bayesian Methods
  - Dempster Shafer Theory
  - Transferable Belief Theory
- Recent Work in Data/Information Fusion
- Overview of Public Safety using Crowd Sensors Initiatives (National Technical Challenge by IRL)

3. TARGET AUDIENCE

This tutorial is designed for students and researchers in Computer Science. Elementary knowledge of text mining is assumed. This topic is expected to be of wide interest given its overlap with data mining, text mining, NLP, Streaming Data and BigData. We plan to give a 3 hour tutorial.

4. SPEAKERS

L Venkata Subramaniam manages the information processing and analytics group at IBM Research India. He received his PhD from IIT Delhi in 1999. His research focuses on unstructured information management, statistical natural language processing, noisy text analytics, text and data mining, information theory, speech and image processing. He often teaches and guides student thesis at IIT Delhi on these topics. He co founded the AND (Analytics for Noisy Unstructured Text Data) workshop series and also co-chaired the first four workshops, 2007-2010. He was guest co-editor of two special issues on Noisy Text Analytics in the International Journal of Document Analysis and

Recognition in 2007 and 2009. He can be reached at lvsubram@in.ibm.com.

Sameep Mehta is researcher in Information Management Group at IBM Research India. He received his MS and Ph.D from The Ohio State University, USA in 2006. He also holds an Adjunct Faculty position at International Institute of Information Technology, New Delhi. Sameep regularly advises MS and PhD students at University of Delhi and IIT Delhi. He regularly delivers Tutorials at COMAD (2009, 2010 and 2011). His current research interest includes Data Mining, Business Analytics, Service Science, Text Mining, and Workforce Optimization. He can be reached at sameepmehta@in.ibm.com.

R E S E A R C H   T R A C K

Connectivity-Tolerant Query Optimization Over Distributed Mobile Repositories[*]

Sharma Chakravarthy, Aditya Telang[†] Mohan Kumar
Mark Linderman[‡] Sanjay Madria[§] Waseem Naqvi[¶]

Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX, USA

sharma@cse.uta.edu, adtelang@in.ibm.com, kumar@cse.uta.edu,
Mark.Linderman@rl.af.mil, madrias@mst.edu, Waseem_Naqvi@raytheon.com

ABSTRACT

Query processing and optimization in centralized and distributed environments is well-researched. Centralized query optimization focused on minimizing the number of input/output (or I/O) from disk. Distributed query processing focused mainly on maximizing local computation and minimizing data transfer between nodes. Here the distribution of data was pre-determined and both connectivity and bandwidth were pre-defined *and* guaranteed. Work on sensor data acquisition deal with non-join queries without taking mobility and connectivity interruptions into consideration. However, these assumptions are no longer true when queries are executed over repositories stored in mobile aerial vehicles which collect, process, and store data in real-time, and connectivity changes significantly over the duration of interest. Currently, only data in one vehicle can be queried by the ground control.

This paper explores query processing and optimization issues along with concomitant metadata needed for processing/optimizing queries over distributed, mobile, connectivity-challenged environments. Since response-time and fault-tolerance are the main focus, we propose plans using join, semi-join, and replication-based approaches. We propose and evaluate several heuristics for this environment ranging from greedy to cumulative approaches along with the use of replicated copies of data. We have performed elaborate experimental analysis to validate heuristics that work well for this environment. As maintaining replication is a challenge in this environment, we summarize our initial approach. This work on connectivity-tolerant query optimization is part of a larger middleware-based, service-oriented architecture.

[*]AFRL has approved this work for public release; distribution unlimited. Case No. 88ABW-2012-5499
[†]IBM Research, Bangalore, India
[‡]Air Force Research Labs, Rome, NY
[§]Missouri University of Science and Technology
[¶]Raytheon Corporation, MA

1. INTRODUCTION

As part of a larger effort on distributed middleware-based architecture for fault-tolerant computing over distributed repositories, we address query processing and optimization in this paper. A brief description of the larger problem is provided for understanding the context for this work.

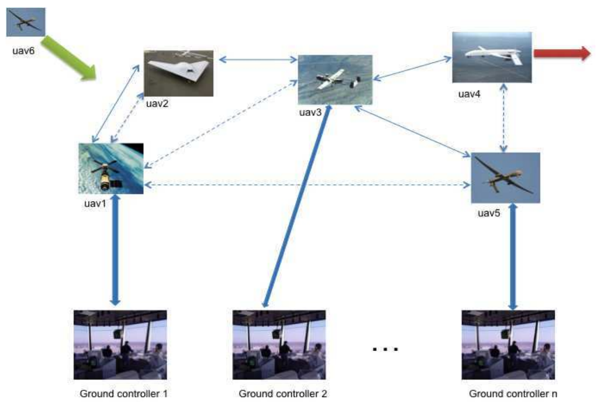


Figure 1: Example of Nodes and Connectivity

The general problem can be stated as follows: Consider a number (2 to 15) of nodes (unmanned aerial or other vehicles termed UAVs in this paper, and ground operators) whose connectivity is dynamically changing, and whose data bandwidth can vary from low to high. In this setting, how do we accomplish a specific task (query, search, subscription notification) that uses data and services from multiple nodes (for computation or collaboration) that are subject to QoS (Quality of Service) requirements (e.g., time to first result, response time). In other words, each node is independently acquiring multiple/different data types (e.g., location, telemetry, and images) and storing them locally. The data is stored in the form of Managed Information Objects (or MIOs) and can be sent on-demand to ground operators, and others nodes based on connectivity. There is also a need for combining (or joining) data from multiple nodes to get a better understanding of the overall situation. Data stored in a node is defined using type, metadata, and the payload. The communication between the nodes is through RF or satcom or other types of links (e.g., Link 16). It is also assumed that the nodes can be of different types based on processing capacity, storage, types of data it can collect, up/down link bandwidths, and latencies of data transfer. Nodes can also play different roles (depending upon the resources available onboard): i) collect data and forward it, ii) collect data, processes it, and forward both collected and processed data, and iii) collect, process, store/hold, and forward data. A single node can play different/multiple roles for different types of data. Their roles may change over time.

A typical scenario consists of a number of Airborne platforms (UAVs, Helos, Fighters, AWACs, etc.) which are traveling at various speeds (100, 200, 500 knots etc.), some in formation and some on independent tracks. Each has an associated *ground platform* that are either stationary or moving. Stationary platforms have semi permanent positions whereas Mobile ones may be on vehicle or foot. The connectivity among all airborne platforms is intermittent based on distance, line-of-sight, obstacles, cloud coverage etc. The transmission bandwidth is different for receiving and sending and depends on a number of factors such as distance, orientation, obstacles along the path etc. Each node (an airborne platform) has storage that is meaningful for the node type. Although computing power varies from node to node, we can assume that it is sufficient to run a local database management system (DBMS). Relational DBMS is assumed. Power is assumed to be a non-issue in this work because these platforms, we were told, have enough juice for the duration of the mission. This general scenario arises in various contexts:

- Disaster management, such as flooding, hurricanes, and evacuation. Information needed: evacuation routes, extent of damage, view of the area affected.
- Cooperative Combat Air Patrol. Mixture of UAVs, manned fighters, and AWACs cooperatively defending a region. Information needed: Signals, Lines of Bearing, contact positions, tracks

The scenario described above is illustrated in Figure 1. It is assumed that ground controllers are always in contact with their respective UAVs. Connectivity of other nodes (or UAVs) depends on dynamic factors. The connectivity (or disruption) of the nodes changes dynamically in this scenario as illustrated by solid lines and broken lines. Each line type (solid or broken) represents a different configuration of the network – one partitioning the nodes into two graphs and the other maintaining reachability for all nodes. The figure also shows new nodes coming into and existing nodes going out of the network.

Currently, it is only possible to process queries on data stored in a single airborne platform. Current state-of-the-art in distributed query processing assumes fixed, hard-wired connectivity among participating nodes. Replication is considered from an availability (of data) viewpoint and not from connectivity viewpoint. Latest work in sensor query processing [17] does not deal with mobile platforms with resident relational DBMS and intermittent connectivity. It is also possible to download data into ground nodes (from all nodes) and then process queries over those nodes. This results in delays that is not acceptable. Also, the data collected by multiple vehicles for a situation provides a holistic view and hence it is important to have the capability to issue queries *in real-time* that can be processed over all the relevant data available in one or more airborne platforms. Based on the requirements of the situations listed earlier (especially response time), it is important to have the capability to process queries over data in multiple nodes as they are being acquired.

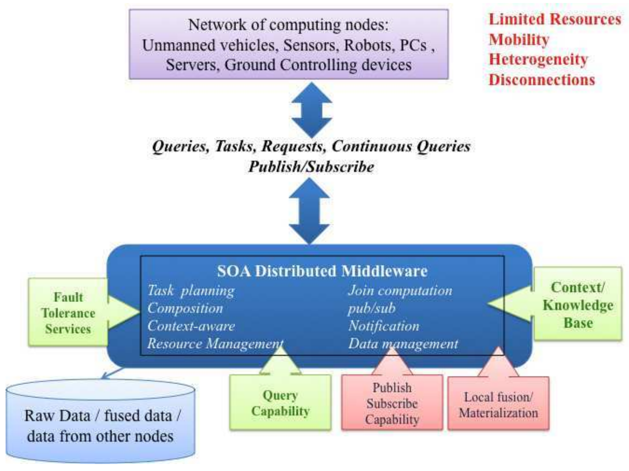

Network of computing nodes: Unmanned vehicles, Sensors, Robots, PCs , Servers, Ground Controlling devices
Limited Resources
Mobility
Heterogeneity
Disconnections
Queries, Tasks, Requests, Continuous Queries
Publish/Subscribe
SOA Distributed Middleware
Task planning
Composition
Context-aware
Resource Management
Join computation
pub/sub
Notification
Data management
Fault Tolerance Services
Context/ Knowledge Base
Raw Data / fused data / data from other nodes
Query Capability
Publish Subscribe Capability
Local fusion/ Materialization

Figure 2: Pluggable Middleware Architecture

The following are examples of queries that need to be executed on networked, distributed information sources.

1. Get all images taken within last 5 minutes of the area bounded by ⟨latitude1, longitude1⟩ and ⟨latitude2, longitude2⟩.
2. Get all SAM (surface to air missile) locations within 12 NM (nautical miles) of the area bounded by ⟨latitude1, longitude1⟩ and ⟨latitude2, longitude2⟩.

Since each node is autonomous and may belong to a group that needs to solve problems collaboratively, there is a need for two fundamental components that form the core of our overall approach:

- a common middleware component that is common to and present in all the nodes and a context (as a knowledge base or KB) that holds the capabilities, network configuration, and
- current state of the network at each node which is managed and used by the middleware.

The KB will also include the global requirements as capabilities of connected nodes are dynamically gathered. The context information can be customized/tailored either to a node, a task or for a set of tasks. The middleware heavily relies on the context to perform operations, knows the capabilities of self and other nodes, and to perform tasks collaboratively. A service oriented architecture (SOA) for the middleware is used to build larger systems in which this fits as a component seamlessly.

Figure 2 shows the service-oriented architecture (SOA) for the middleware for supporting query processing (and other services) over distributed repositories and accommodate fault tolerance. The overall architecture includes a middleware in each node that has a number of services (based on SOA) for collecting, managing, replicating data and metadata for the purposes of routing and query processing. Each node will have the SOA middleware and as many plugin components as needed. As a node collects data, it is stored in the repository on that node. Connectivity and replication information is periodically exchanged between nodes and stored in the context/kowledge base. For details on other services, please refer to [7] which contains an accessible url.

Contributions: Some of the key contributions of this paper are as follows:

1. Formulation of the distributed query optimization problems for ad-hoc connectivity in the presence of connectivity interruptions,
2. A cost metric that is different from traditional distributed cost metrics,
3. A query processing strategy with partial independent computations in different nodes, and
4. Replication of data for dealing with availability and incorporating it into query optimization

Overall, the novelty is in generalizing distributed query optimization to ad-hoc networks with mobile platforms, intermittent connectivity, and replication.

The remainder of the paper is organized as follows. Section 2 defines the problem being addressed in this paper. Section 3 discusses related work on query processing and metadata management. Section 4 discusses meta data used for query processing and its management. Section 5 briefly summarizes our replication strategy. In Section 6, we discuss our approach for processing queries, plan generation alternatives, and introduce heuristics appropriate for this environment. Section 7 has elaborate experimental analysis and their interpretation. Section 9 has conclusions.

2. PROBLEM STATEMENT

The focus of this paper is on processing SQL queries over distributed repositories collected/stored on each vehicle with the specified constraints on connectivity and reachability. This will allow for holistic queries without even having to know which repository contains what information. Although we are using SQL queries referring to relations in a node in this paper, a GUI can easily generate these queries from an interactive interface. It is meaningful to assume that each node has secondary storage of reasonable size. It is also assumed that each node has enough computing power to support a database management system (DBMS) that can process queries from its local (secondary) storage. A relational DBMS is assumed at each node.

The problem at hand is somewhat different from the traditional query processing that has been developed for centralized, distributed, and federated architectures. Although the computations/operators (e.g., join, semijoin) are the same as that of traditional query processing systems, the environment and the goals of these computations are quite different. Instead of knowing the schema, the data is published using a managed information object (or MIO) that needs to be used efficiently. An MIO (used to represent/encapsulate data) consists of: data type, metadata, and the payload. The metadata could be as simple as schema information or it can consist of additional information, such as range values, organization of data, number and types of objects in a picture etc.

Another difference is the need for replication of data – not from the viewpoint of local processing, but from the viewpoint of accessibility or reachability. Compared with earlier approaches where the nodes at which data was stored (or even replicated) were pre-determined, in the current scenario it is an important decision that has to be made dynamically by the system. As connectivity is not complete, multiple hops may be needed to reach a copy of the data. Furthermore, nodes can even store data that may not be directly useful to that node but is in close proximity for others that need it. When a node moves away (i.e., is not a neighbor anymore), there is a need to decide whether to keep the copy in that node or not. The utility of data and its copies need to be optimized using some metric (or a combination) such as cost of storage, cost of communication, time for data transfer, and longevity of storage.

In this paper, we address the problems of: query plan generation for this environment, relevant heuristics that are meaningful for this architecture, and use of replication for improving query processing. A prototype implementation developed in Java is used for extensive experimental results that validate our approaches and inferences.

3. RELATED WORK

Traditional relational database management systems (DBMSs), consisting of a set of persistent relations, a set of well-defined operations, and highly optimized query processing and transaction management components, have been researched for over several decades and are used for a wide range of applications. Typically, data processed by a DBMS is less frequently updated, and a snapshot of the database is used for processing queries. Abstractions derived from the applications for which a DBMS [5, 1, 15, 12, 18] is intended, such as consistency, concurrency, recovery, and optimization have received a lot of attention.

Query processing is a key consideration in database management systems. For this reason, query optimization has been one of the most active research areas since the advent of relational DBMSs. The acceptance and success of relational systems can be attributed largely to advances in query optimization over several decades [19, 11]. A major advantage of relational systems over earlier technologies is that the users of a relational DBMS are relieved of the need to describe their queries procedurally. More important, users are not required to understand the details of physical representation and its impact on queries posed to the DBMS.

In a distributed (or even a multi-database) environment, queries are decomposed, and query fragments are directed to particular sites (or databases) for processing [3, 2, 16]. Distribution of the database reduces the size of the data stored at each node, increases the locality of reference for the queries processed at a given node. Replicated databases provide an additional opportunity – that of choosing the site (at which a subquery is sent for processing) to increase the probability of overlap with other subqueries. Hence, queries processed at a site may have a lot of overlap of the data they access.

Other forms of query optimization, such as semantic query optimization [9], multiple query optimization [8, 20], and more recently, continuous query processing [6] have focused on modeling, scheduling, and load shedding strategies. The work presented in this paper is related, but is distinctly different from them. In this work, queries are not transformed using semantics, multiple queries are not batched and optimized, and continuous query processing techniques deal with a different set of metrics and their optimization is very different from what is required for this scenario.

Several middleware architectures have been developed in the recent past to support mobile ad hoc networks (MANETs), sensor networks, and pervasive systems. Boulkenafed and Issarny develop a comprehensive middleware for data sharing in MANETs [4]. The focus of the work however is minimizing energy consumption. Kalasapur et al. developed an elegant middleware for service provisioning in pervasive systems with mobile nodes [13]. Tamhane and Kumar have developed a resource management mechanism for pervasive systems with underlying ad hoc networks [21]. None of these works consider dynamic networks such as that of UAVs, where node mobility is a regular feature rather than a rarity. Christman and Johnson discuss a customized self configuring architecture designed for UAVs [10]. However, they do not deal with on content sharing and query processing. The middleware architecture proposed in [14] attempts to address this important issue in UAV based networks.

4. METADATA AND ITS MANAGEMENT

In order to process queries, minimal information about the schema, connectivity of the nodes, replication information (if any) as well as available cardinality and other statistics need to be available in each node. Furthermore, some of the above need to be kept current in this dynamic environment. At the core of our middleware is the use of graph theoretic and sub-graph matching techniques to ensure network status awareness and data access. A graph structure is created to capture the essence of data objects/services, corresponding computing nodes and the relationship among the data objects as well as the nodes. The associated middleware tools facilitate the response to queries in dynamic heterogeneous environment comprising mobile nodes. The proposed service provisioning framework is flexible in representing metadata and services, and adaptive to changing environments by incorporating the replicated copies. We assume the following information in the form of tables accessible to the local database.

Data at each node is assumed to be a relation with the schema shown in Table 1. R_{ij} corresponds to relation R_i at node j. R_{ii} ($i = j$) will be used to represent the *primary copy* of a relation at node i. R_{ij} (i <> j) will be used to indicate the replica of R_i in node j. A field 'TimeOfUpdate' is maintained for each update that happens over the Meta Data to estimate the accuracy of data and keep a track of how recently the update has been done.

A number of additional information about the characteristics of each R_{ii} is maintained in a node i (and periodically propagated to all other nodes) for the purpose of query plan generation and cost estimation. If a relation R_i is replicated at this node (j), then for each replicated relation R_{ij}, we need to maintain the same information as in Table 1. The difference is that this information may not be current. Every node maintains a copy of its original relation that is stored at some other node. Currently, replication is assumed to be a single copy and complete for each relation. Network Managed data is maintained and updated by the middle-ware, and accessed for processing by the local query processor for executing intermediate steps of a query plan.

Selectivity for simple and composite conditions are calculated using standard formulas [19, 16] based on the information in Table 2.

A Relation-to-Node mapping table, as shown in the Table 3, is maintained by the message management system at each node which indicates the location of the original and the replica of a Relation. A value of 0 in the replica node column indicates that the replica is not complete at this point in time and hence is not considered for generating a query plan.

Name	Original Node	Replica Node
R1	1	4
R2	2	1
...	...	...
Rn	N	k

Table 3: Relation and Replica Locations

Finally, a Connectivity map is maintained at each node which checks for the existence of a connection between any two nodes and the corresponding bandwidth between them. If the Received Signal Strength (RSS) is zero or below a threshold, then the connection is considered to be 0 and 1 (or present) otherwise. RSS value lies on a scale of 1 to 10. The actual RSS value is used in cost estimation. LSF (Link stability Factor) is a function of rate of change of RSS value over a period of time. LSF, to some extent, measures the stability of the link over a period of time. This is important as the plan is generated once and the execution of steps take some time. A pair is considered for the plan generation if the RSS value at the instant is 1. A sample connectivity map is shown in Table 4. *Note that bi-directional connectivity is maintained as the bandwidth is different between uplink and downlink.* See [14] for network related issues.

5. REPLICATION STRATEGY

In order to ensure accessibility and fault-tolerance, each data object is replicated on other nodes. Currently, there exists only one replica of a given data item. N_s represents the source node, where the original copy of data item D_i was acquired. N_c is the candidate node that will contain a replica of data object D_i. When N_s decides to replicate its contents on another node N_c, a node from the set of the nodes that are immediate neighbors of the source node is selected as candidate nodes for replication. Immediate neighbors are those nodes which are directly connected to the source node. The source node tries to replicate all its tuples on the chosen candidate node. For each of the above selected candidate nodes, a cost function $C(s, c)$ is computed. The node with the lowest cost is selected as a candidate for replication. The cost function to determine the candidate node for replication is dependent on the following factors: *Bandwidth* defines the closeness of N_c from N_s in terms of bandwidth. Greater bandwidth is desirable; *Linkstability* is a measure of stability of the link between nodes N_s and N_c. Greater stability of the link between the two nodes implies better longevity; and greater $Degreeofthenode N_c$ indicates better accessibility of replicated data. Additional details can be found in [14].

6. QUERY PROCESSING AND PLAN GENERATION

Although it is tempting to try to optimize a query from scratch as is done traditionally, we need to take the environment and constraints into account for proposing an appropriate solution. The focus here is to generate a query plan that can complete the execution of a query with minimal data transmission cost and good response time. Hence, a plan generator that tries minimize I/O in each node is not the best way as the local DBMS is likely to do a better job; and we need to leverage that. Hence, we decided to delegate local optimization to the DBMS at each node and concentrate on a plan that minimizes data transfer (or data movement) for processing a query. As a result, a query plan for this scenario is envisioned as numbered sequence of plan steps that can be easily interpreted and executed at any node[1]. Table 5 gives a description of a plan format. Each step includes the operation to be applied, the data items involved, the node where it is applied, the name of the result and the node where it is created.

Unlike traditional query processing, the plan needs to be sent from node to node[2] (or partial plans generated at each node which is not considered in this paper) for the purposes of query processing. A counter, as part of each plan, indicates the next step to be executed and is initialized to 1. An example of a query plan is shown in Table 6.

The plan format described above is sufficient to describe any arbitrary relational query plan involving selects, projects, and joins (also known as an SPJ query). The above format can also accommodate SQL aggregate operators, such as a SUM, COUNT, AVERAGE, MINIMUM, and MAXIMUM. A query is executed as follows. A complete plan is generated at the node where the query is received using the metadata stored in that node. The plan is then sent to the node in which the first operation takes place (if it is different from the node where the query plan is generated) along with the plan step counter. The interpreter in that node uses the plan step counter to execute as many steps as possible in that node. When a move or copy is encountered, it sends the data as well as the plan (actually the remaining portion of the plan to reduce the amount of data transferred) to the next node. This process continues until the last step of the plan is executed. The result of the query will always be sent to the node at which the query was received.

Currently, a complete query plan is generated as follows. Each node in the architecture has the same query plan generator and uses *only the Metadata in that node*. Note that the metadata is updated by the underlying mechanism briefly indicated in Section 4. The query plan is constructed one join/semijoin at a time. Costs of partial plans are computed using well-defined statistics and formulae for computing selectivities for conditions and join. The lowest total cost query plan is used as the final plan after the plan space is explored either exhaustively or using heuristics. This will result in a good plan (or an optimal plan). Several heuristics are explored as part of this project to reduce the total computation required for generating a plan and still generate a good plan[3]. These heuristics are compared experimentally with respect to replication and connectivity scenarios.

The complexity of the optimal plan generation is k^n where n is the number of joins and k is the number of alternatives for each join. Currently, k being used is 18 (three alternatives for join, semijoin, & hybrid alternatives, and the same using replica as well). Note that this is at the logical level. For each logical join alternative, there will be many physical alternatives making the plan space significantly larger. Assuming three joins, we need to explore 5000+ alternative query plans and compute cost for each one of them. For plans with more than three joins, this exhaustive approach is not viable. Hence, we have incorporated some heuristics to limit the number of plans generated by pruning plans carried forward after each join. A query optimizer has been implemented to validate the heuristics and their effectiveness on synthetic data and multi-join queries that simulate actual data sets.

Cost for our plans is mainly data transfer cost which in turn depends on the width of the tuple and cardinality of the relation (intermediate or otherwise). Hence it is important to estimate the number of tuples as well as their width. Statistics in the form of cardinality and domain characteristics are used for this purpose. Join and condition selectivity are inferred from the statistics maintained. Intermediate result sizes are also estimated as its accuracy is important as the choice of the best query plan is primarily based on the cost of data transfer based on availability of connectivity. The statistics used for evaluating the cost of a (partial) query plan is the same as the ones used in traditional and distributed query processing [19, 16]. All of these are well-established for the relational model. We do not include the processing cost for the operation/plan, but only the data transfer cost. Processing cost depends upon the availability of index and other structures and mainly influences the order of join (which we take into account in our plan generation process). As future work, it will be useful to explore what access structures are meaningful and take the processing cost into account as well. In each node, the plan can be executed by converting it into an SQL statement if a relational database is used for storing data in that node.

To improve the accuracy of selectivity, for each attribute of R_{ii} on which a condition has been applied, selectivity information is maintained as follows. Table 7 reflects the **actual** selectivity values for conditions on that relation and will be used when the same or similar condition is encountered in a later query. Otherwise, selectivity formulas are used for calculating the resulting relation cardinality. The conditions are maintained at the component level using a hash table which can be associatively searched using the relation and condition. The intermediate relation cardinality and width are also maintained.

[1]In fact, we assume that at each node, plan steps are combined to generate an SQL query to be processed locally accessing *only* local data.

[2]As an alternative, it is possible to simultaneously send the entire plan or preferably portions of the relevant plan steps to each node. If this alternative is used, a synchronization mechanism is needed to execute plan steps in the correct sequence without any need to transfer plans. It is also possible to dynamically generate plan steps at each node when needed rather than generating the entire plan to start with.

[3]Note that, in general, the objective of query optimization is not as much as generating an optimal plan by spending a lot of resources, but to certainly avoid *bad* plans and do it fast.

Relation	C1	C2	C3
R1	0.2	0.5	
R2		0.6	0.67
R1	0.9	0.1	0.7

Table 7: Selectivity Table

6.1 Plan Generation Implementation

The query plan generator is implemented in Java. A relational database is used for storing metadata (as will be done in each node). A constants Java file is used for conducting experiments and to setup parameters for varying connectivity and replica information (as shown in Figure 3). An interactive option is also available to input query, load metadata from a file, and analyze individually best, worst, or any plan generated. For details of implementation refer to [7].

The generator begins by generating all distinct partial plans (from an initial empty set) for each join. As an exhaustive algorithm, it generates 18^n plans for a query containing n joins. It is evident that this approach is not viable beyond a few joins. This is being done so that we can compare heuristics-based plans with the optimal ones to analyze the effectiveness of heuristics we come up with (e.g., top-k in each iteration, top-k cumulatively, top-k for each type of plan.) for queries with fewer joins. The generator then iterates through the relation list and creates the necessary plan steps. Then all of the attributes required are projected on the output and join condition attributes to minimize the data transfer across nodes which form the bulk of the cost of query processing in this environment. Since most of the plans will use these initial select or project statements (to reduce the width and cardinality of the relation), these same statements are attached to every plan. For plan alternatives using joins the generator moves the required relations to the location of the join and then performs the join. Even for this, projections are applied to reduce the overall width and cardinality of relations moved. The plan class takes care of updating intermediary name, location, and condition information. Then the generator moves on to the next plan.

For plan alternatives using semijoins, the relation that will be semijoined to is copied and projected on the attributes used in the specific join condition to minimize data transfer. Then it is moved to the location of the semijoin. The semijoin is performed. When the semijoin step is added to a plan the plan updates name, location, and condition information and in the case of semijoins the output relation and the relation that still needs to be semijoined to finish the operation is added to a stack to keep track of the remaining semijoins (to generate chained semi join plans). Note that a join can be processed as a sequence of two semijoins. However, when multiple semijoins are performed in a sequence, the second semijoin needs to be performed in reverse order (hence a stack). The next plan is then processed. For multiple joins, after all of the plans have been processed with the first join, all of the joined relations will be projected on the remaining join attributes required and then algorithm will iterate through all the plans again performing the remaining joins and semijoins. After a relation has been joined, its current location is considered to be that of the result of the join even if it currently has a replica, which may cause some of the plans to be the same. After all cases have been exhausted, the algorithm goes through and finishes each case by iterating through the stack of remaining semijoins completing the remaining semijoins in reverse order and then moves the final relation to its output node. During each step of the plan generation, the cost associated with a move or a copy is calculated, if there is no direct connection between nodes the cost is considered prohibitively high and value is automatically forced to a very high level by using a very low bandwidth for the calculation. After calculation the plans can be viewed in sorted form. The plan generator generates a summary of: number of plans generated, lowest and highest cost plan numbers. It is possible to view any of the plans in detail. The same process is used for generating plans using heuristics except that a subset of plans are used in each iteration which are selected based on the specific heuristic.

The plan generator also includes a network component that generates the connectivity matrix using the seed provided. Each element in the matrix represents the cost of the link from node x to node y. Number of connections is also specified as part of the configuration. The connectivity matrix generated is consistent with the bandwidth assumptions for this scenario. The connectivity matrix is updated to simulate movements of the nodes.

6.1.1 Sample Best and Worst Plans

Consider a multi-join query that is sent to node 1 and the results expected back in node 1.

```
Node        TARGET 1
SELECT      *
FROM        U_1_D,U_2_D,U_5_D
WHERE       ((U_5_D.OBJTYPE=1))
        AND ((U_1_D.LAT>U_2_D.LAT))
        AND ((U_2_D.LONG>U_5_D.LONG));
```

Plan Number	Total Cost (in milli secs)	Remarks
253	**493.873**	alternatives 15 (first join) and 2 (second join)
263	585.942	alternatives 15 and 11 (only semijoins)
3	*175117.8*	alternatives 1 and 3

Table 8: Sample Plan Costs

Below, we present Lowest cost, semijoin only cost, and highest cost plans for the above query in Table 8 and additional information about how they were generated in terms of plan combinations for a network configuration. In the above the best plan seems to be a combination of join and semijoin. The worst plan seems to be made of only joins. As can be seen, the difference between the best and the worst plan is significantly large. Hence, it is important to choose plans closer to the *best* plan (i.e., a *good* plan).

6.2 Heuristics-Based plan generation

The purpose of generating an exhaustive plan space as indicated above is to demonstrate the cost differences between the best and the worst plans. The above algorithm is still not exhaustive in that it does not consider all possible join combinations. As can be seen clearly, there is a significant difference between the best and the worst plan. The goal of query optimization is not necessarily to choose the *optimal* plan, but to avoid bad plans and choose a *good* (closer to the optimal and far from the worst) plan.

During testing, we also realized that the connectivity plays a critical role in that if only one way connection is available between nodes, it impairs good plan generation as semijoin-based plans need to finish the second half of join by bringing the results back to that node. In order to generate a plan without exhaustive search of the plan space, we have proposed a number of heuristics to the above algorithm to compare their performance with the optimal plan. We use this prototype implementation to analyze various aspects such as connectivity, bandwidth, as well as selectivity to understand the types of plans generated and the effect of these parameters on total plan cost. We have identified the following heuristics to be useful and have implemented them so that we can compare them to the optimal ones to determine when and which heuristics to use for queries with more joins.

1. **Top-k Iteration:** Plan generation is iterative with respect to joins. For this heuristic, we choose top k (where k can be specified as a parameter) *lowest* cost partial plans in each round of expansion or iteration. Note that each iteration in our approach corresponds to processing a join. The number of iterations is equal to the number of joins. The intuition behind this approach is to use a greedy local selection and hope that it will turn out to be good globally as well. This significantly reduces the size of the explored plan space.

2. **Top-k Cumulative:** For this heuristic, we choose top k lowest *cumulative* cost (up to that point) plans in each round/iteration of expansion. Again, the intuition is that the cumulative cost up to this point is more meaningful (than Top-k-iteration, for example) and this would lead toward a good overall plan. Note that this and the above heuristic will be identical up to two joins. We expect this heuristic to do better than the previous one as the number of joins increase.

3. **Top-k Join-type:** For this heuristic, we categorize plans into join-based, semijoin-based, and hybrid (a combination of join and semijoin). we choose top k *lowest* cost plan from each category for expansion in each round. The Top-k join-type is a different type of heuristic as we have different types of partial plans and their costs are likely to be different. Here, k lowest cost plans from each type is chosen for the next round. In order to compare them in a fair manner, the k value need to be lower (1/3 as we have 3 join types) so that the same number of plans are carried forward in each round. Otherwise, this approach is likely to explore a larger plan space and do better than the other two heuristics.

In addition to the above, a number of other possibilities for plan generation exist: i) incremental plan generation, ii) looking ahead at connectivity and pruning plan alternatives, iii) getting dynamic cost information and then generating partial plans

Note that connectivity, in this context, is likely to play a significant role not only in the generation of a complete plan, but also its cost. If sufficient connectivity does not exist among the nodes that participate in the query (including the nodes that have a replica), a complete query plan may not even be feasible. The presence of replica increases the probability of generating a complete plan and if several exist, heuristics hopefully will choose a good one without having to generate all plans. A heuristic that incorporates connectivity would be very useful for this environment.

The above three heuristics have been implemented in our prototype. The software has two modes: interactive and experimental to make it easy to test and use. In the interactive mode, a query can be given at the prompt (or in a file) and a heuristic specified for its plan generation. The generator will indicate the number of plans generated as well as the lowest and highest cost plans (along with plan number). One can output (or look at) any plan in details by typing the plan number. It is also possible to provide a file input to process multiple queries in this mode. The selectivity and cardinality information is statically initialized. The connectivity is also initialized at the start of the system. This can be easily changed by loading a new or different relations and connectivity information before executing the plan generator.

In the experimental mode, the configuration is set using a Java Constants class (a sample is shown in Figure 3. The input consists of: number of queries to be generated, seed for query generation, number of connectivity configurations to be used in the experiment, seed for configuration generation, and connectivity factor. The generator has a random query generator on the schema stored in the system and generates the desired number of queries for which minimum and maximum number of joins can be specified. The seed is to ensure repeatability of experiments as well as generate a new sequence of pseudo-random queries. The same is true for network configurations and its seed. The connectivity factor is use to control the sparseness of the connectivity matrix. If there are n nodes, the connectivity factor can vary from 0 to (n-1), 0 indicating no connectivity at all and (n-1) indicating complete connectivity. The connectivity itself is generated randomly to satisfy the parameters specified.

The above setup allows one to perform different types of experiments. For each query, connectivity can be changed to determine how the plan cost changes and can also compare the optimal cost with heuristics-based plan costs. It is possible that due to the connectivity, a number of plans cannot be completed resulting in a high cost. Queries or connectivity sequences can be changed, independently, by varying the corresponding seed.

7. EXPERIMENTAL ANALYSIS

In order to test the effectiveness of the heuristics proposed for the query plan generator, we performed several experiments using these heuristics across different connectivity matrices and several different queries. The following Java interface (see Figure 3) was typically used for setting up parameters for all experiments.

A sample set of queries used for experimentation is shown in Figure 4. Two and three join queries with different selection and join conditions have been purposely chosen so that they can be compared with optimal results. This will force the execution of plan in multiple nodes and also brings in the use of replicated relations based on the connectivity. These queries were generated by domain experts who have experience in these scenarios. Finally, the cardinality for all relations ranges from 100000 tuples to 505000 tuples. This cardinality also represents the amount of data acquired during a mission. *We have presented tables instead of plots as the range of values from our experiments is quite large and* *hence is not conducive to plotting.*

7.1 Comparison of Heuristics

In this experiment, we tested the five queries (shown in Figure 4) and tested the three heuristics along with the optimal algorithm on the same configuration of the connectivity matrix. Table 9, shows the cost (in milliseconds) incurred by the various approaches towards generating the top-3 best plans (average) for 5 different queries on the same connectivity matrix configuration. Based on the results, it can be observed that the plan generation process depends hugely on the connectivity between the nodes. Among optimal plans, the semijoin ones seem to perform better as expected since data transfer is reduced significantly. Further, amongst the different heuristic approaches, the semijoin approaches (either top-k-iterative or top-k-cumulative) appear to perform better for the current set of connectivity configurations.

7.2 Plans costs with and without replication

In this experiment, we compare the costs of generated plans with and without replication to establish the need for replication and its importance for this environment. In order to test in the *absence* of replication, we selected each query independently and altered the settings of Table 3 such that for the nodes involved in the corresponding query no replica existed. For instance, considering *Query 2*, we modified Table 3 such that replicas for nodes 5 and 10 did not exist in any other node. We then evaluated the optimal as well as heuristic-based plans for each query in the absence of replication, by averaging the costs obtained from the corresponding top-3 plans. We then enabled replication by creating replicas of the nodes, as shown in Table 3, and evaluated the costs in the presence of replication.

Table 11 shows for a specific query (*Query 3*) the costs obtained by each plan in the presence and absence of replication. It is clear that the processing cost without replication is significantly higher (as high as 6 times). We also wanted to understand the behavior of averages. Table 12 displays the average costs obtained across all five queries, when replication was present and absent. We observed that, in the absence of replication, it was difficult to obtain a low cost plan (due to the nature of the connectivity between the different nodes); as a result, a relatively high-cost plan has to be selected. In contrast, replication provides a distinct advantage as a low cost plan, involving the replica nodes can be obtained even though the connectivity between actual nodes involved in the query may not exist. Consequently, the presence of replication yields comparatively low-cost plans, and hence proves to be fruitful in such scenarios where the connectivity between nodes is dynamic and susceptible to frequent changes. This is for a single copy replication. It would be interesting to study the tarde-offs between number of copies and plan costs.

Method	Replication	No Replication
Optimal Join	63.76	175.73
Optimal semijoin	135.33	326.28
Top-K Cumulative Join	85.57	195.14
Top-K Cumulative semijoin	78.46	179.07
Top-K Iterative Join	70.76	379.54
Top-K Iterative semijoin	129.39	894.21
Top-K Join-type Join	80.76	391.72
Top-K Join-type semijoin	129.39	666.67

Table 11: Replication Vs. No Replication: Effect on costs for Query 3

Method	Replication	NO Replication
Optimal Join	8.91	29.41
Optimal semijoin	29.33	126.81
Top-K Cumulative Join	527.21	143.73
Top-K Cumulative semijoin	279.21	795.41
Top-K Iterative Join	33.11	177.14
Top-K Iterative semijoin	318.25	828.21
Top-K Join-type Join	801.49	935.72
Top-K Join-type semijoin	304.71	899.67

Table 12: Replication Vs. No Replication: Effect on costs across all queries

7.3 Impact of Connectivity on Plan Cost

In this experiment, we present a single query (shown below) and computed the top-3 plan cost using the heuristics proposed along with the optimal plan cost on six different configurations of the connectivity matrix. We had to keep the connectivity large; otherwise, no (or not many) plans were generated. Since the connectivity matrix is large in size, we do not show it here. Instead, we have displayed a sample configuration file earlier. We have done this experiment on several queries with similar results.

```
Query 1   target 2
SELECT    * FROM UAV_2_DATA, UAV_4_DATA, UAV_6_DATA
WHERE     ((UAV_2_DATA.NODEID=76)) AND ((UAV_2_DATA.LONG>=804))
          AND ((UAV_6_DATA.LONG<=540) AND
          ((UAV_2_DATA.LAT=UAV_4_DATA.LAT)) AND (UAV_4_DATA.LONG=UAV_6_DATA.LONG));
```

Table 10 shows the cost (in milliseconds) incurred by the various approaches towards generating the top-3 best plans for the given query. Based on the results, it can be observed that the plan generation process depends heavily on the connectivity between nodes. For many network configurations, no plan is generated even in optimal join case. However, amongst the different heuristics, the semijoin approaches (both iterative and cumulative) appear to do better and very close to optimal for the current set of connectivity configurations. However, determining the exact relationship between the type of join and the corresponding costs of plan generation will require further analysis and is beyond the scope of this paper.

7.4 Desiderata

It is very clear from the experiments that the proposed heuristics are meaningful and generate good plans that are not too far from the optimal without exploring the entire plan space. The presence and absence of replication makes a significant difference both for the number of plans available and the cost of the plan. This is only for directly connected replica. If multiple hops are included, reachability will be even better (at the cost of transmission cost). Connectivity of the network certainly plays a central role and more attention needs to be placed on heuristics and optimization to include predicted stability of network and its leveraging. Alternate plan precessing strategies will also be beneficial for this environment. As an example, parallel execution of plan steps in different nodes is likely to reduce response time substantially.

8. ACKNOWLEDGEMENTS

Authors would like to acknowledge the support from Air Force Research Laboratory (AFRL) for this work. The work presented in this paper is partially supported by Air Force Research Laboratory (AFRL) grant. Authors would also like to acknowledge the contributions of Danny Hua, Nick Steffen, and Chance Eary on the implementation of the query optimizer prototype used for experimental analysis and all students who worked on this project from all participating institutions.

9. CONCLUSIONS AND FUTURE WORK

In this paper, we have explored SQL query processing and optimization in distributed environments where connectivity is changing rapidly. Instead of optimizing the query from scratch, we have relied on local optimization and have used an incremental plan generation approach with several heuristics for processing a query at the granularity of joins and semijoins and concomitant data transfers. Replicated copies are assumed and taken into account in order to alleviate availability of data due to connectivity issues and increase the probability of an available copy during query processing.

A number of extensions are currently being investigated: i) optimum number of replicated copies instead of a single copy, ii) generating the query plan incrementally and dynamically (due to connectivity issues), iii) use of parallel plan evaluation with concomitant complexity to plan generation and evaluation, and iv) various QoS issues pertaining to query results.

10. REFERENCES

[1] Abraham Silberschatz and Henry F. Korth and S. Sudarshan, *Database System Concepts, 3rd Edition.* McGraw-Hill Book Company, 1997.

[2] P. A. Bernstein and N. Goodman, "The theory of semi-joins," Computer Corporation of America, Tech. Rep. Tech Report CCA-79-27, 1979.

[3] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. Rothnie, "Query processing in systems for distributed databases (SDD-1)," *ACM TODS*, vol. 6, no. 4, pp. 602–625, Dec 1981.

[4] M. Boulkenafed and V. Issarny, "Middleware service for mobile ad hoc data sharing, enhancing and data availability," in *ACM Middleware*, vol. 2672, no. 1. LNCS, 2003, pp. 6–25.

[5] C. J. Date, *An Introduction to Database Systems, Volume 2, Sixth Edition.* Addison-Wesley, Reading, 1995.

[6] S. Chakravarthy and Q. Jiang, *Principles of Stream Data Management.* Springer, 2008.

[7] S. Chakravarthy, M. Kumar, S. madria, and W. Naqvi, "A Distributed Middleware-Based Architecture for Fault-Tolerant Computing Over Distributed Repositories," *TR CSE-2011-8, UT Arlington*, Dec 2011, http://www.cse.uta.edu/research/publications/Downloads/CSE-2011-8.pdf.

[8] S. Chakravarthy, "Divide and Conquer: A Basis for Augmenting a Conventional Query Optimizer with Multiple Query Proceesing Capabilities," in *ICDE*, 1991, pp. 482–490.

[9] U. S. Chakravarthy, J. Grant, and J. Minker, "Logic-Based Approach to Semantic Query Optimization," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 162–207, 1990.

[10] H. C. Christmann and E. N. Johnson, "Design and implementation of a self-configuring ad-hoc network for unmanned aerial systems," in *AIAA*, 2007.

[11] G. Graefe, "Query evaluation techniques for large databases," *Computing Surveys*, vol. 25, no. 2, pp. 73–170, Jun. 1993, (Survey Article).

[12] J. D. Ullman, *Principles of Database and Knowledge-Base Systems, Vol. II.* Computer Science Press International, Inc., MD 20850, 1989.

[13] S. Kalasapur, M. Kumar, and B. Shirazi, "Dynamic service composition in pervasive computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 907–918, July 2007.

[14] M. Kumar, M. L. Sharma Chakravarthy, Sanjay Madria, and W. Naqvi, "Middleware for Supporting Content Sharing in Dynamic Networks," in *MilCom2011, The Military Communication Conference*, November 2011.

[15] M. Stonebraker, Ed., *Readings in Database Systems.* Morgan Kaufman Inc., 1988.

[16] M. T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[17] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, Mar. 2005. [Online]. Available: http://doi.acm.org/10.1145/1061318.1061322

[18] R. Ramakrishnan, *Database Management Systems.* WCB/McGraw-Hill, 1998.

[19] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access path selection in a relational database management system," *Proc. of ACM SIGMOD Conference*, pp. 23–34, Jun. 1979.

[20] T. K. Sellis, "Multiple -query optimization," *ACM TODS*, vol. 13, no. 1, 1988.

[21] S. Tamhane and M. Kumar, "Middleware for decentralised fault tolerant service execution using replication in pervasive systems," in *IEEE PerCom, Sixth International Workshop on Middleware Support for Pervasive Computing*, March 2010.

Timestamp	Nodeid	Lat	Long	Obj_type	Obj_desc	Object_ptr
8 bytes	4 bytes	4 bytes	4 bytes	8 chars	Varchar (64)	Pointer (8 bytes)

Table 1: Relation Format

Attr Name	Type	Cardinality	Position	Width	Min Value	Max Value	Unique values in the range
Timestamp	number	1200	1	100	50	140	90
Lat	number	1200	2	4	10	100	90
ObjType	varchar	4000	3	64	20	350	330
ObjPtr	categorical	2000	3	8	Null	Null	10

Table 2: Relation Metadata

Nodei	Nodej	RSS	LSF	Bandwidth	Start-up Cost
1	3	1	5	100	10

Table 4: Connectivity map

Operation n	Parameter	Operand-1	Operand-1 Loc	Operand-2	Operand-2 Loc	Result Name	Result Loc

Table 5: Plan Format

Operation	Param	Operand1	Operand1 Loc	Operand2	Operand 2 Loc	Result Name	Result Loc
Select	A > 100	R1	1	Null	Null	R1’	1
Project	A1, A3, A4	R1’	1	Null	Null	R1”	1
Move or copy	Null	R1”	1	Null	Null	R”	2
Semi Join	A > C	R”	2	R2	2	SR1	2
Join	B = D	R12	2	R2”	2	JR1	2

Table 6: Example Query Plan

```
Configuration File:
package afrl;
public interface afrlConstants {

int     NUMBER_OF_QUERIES     = 1; //queries generated
String  FILE_NAME             = "outputFiles/apr13_queries_exp1.txt"; // file name
String  NETWORK_FILE_NAME     = "outputFiles/network/apr13_network_exp1"; //conn matrix file
int     SEED                  = 4406235; //for query generator
int     NETWORK_DEGREE        = 11; //# of connected nodes
int     NUM_NETWORKS          = 6; //# of connectivity matric to be generated
int     NETWORK_SEED          = 33152035; //seed for connection matrix generator
int     NUM_NODES             = 13; //# of nodes in connection matrix
int     TopKOptimal           = 3; // optimal plans to display; 0 (all)
int     TopKCumulativeCost    = 3; //carry K plans, 0 (not use this heuristic)
int     TopKIterationCost     = 3; //carry K plans, 0 (not use this heuristic)
int     TopKJoinType          = 9; //carry k number of join type heuristic applying
                                   //cumulative heuristic to k/3 of each type
boolean displayNonConnective  = false; //true to display non connective plans
boolean heuristicDebug        = false; //true to dump heuristic execution data to files}
```

Figure 3: A Sample Configuration Specification

Method	Query 1	Query 2	Query 3	Query 4	Query 5
Optimal Join	8.19	55.34	63.76	2.89	2.52
Optimal semijoin	3.29	8.61	135.33	4.74	2.17
Top-K Cumulative Join	20.54	68.07	85.57	62.78	4.70
Top-K Cumulative semijoin	9.91	39.31	78.46	12.21	4.29
Top-K Iterative Join	8.19	59.54	70.76	5.34	5.62
Top-K Iterative semijoin	4.19	9.31	129.39	162.62	4.95
Top-K Join-type Join	174.20	485.37	80.76	7.12	7.70
Top-K Join-type semijoin	11.91	390.31	129.39	3.21	3.29

Table 9: Heuristics Vs. Optimal: Costs incurred across top-3 plans

```
Query 1: target 2
SELECT   *
FROM     UAV_2_DATA, UAV_4_DATA, UAV_5_DATA
WHERE    ((UAV_2_DATA.NODEID=66)) AND((UAV_2_DATA.LONG>=614)) AND ((UAV_5_DATA.NODEID=77))
         AND ((UAV_2_DATA.LAT=UAV_4_DATA.LAT)) AND ((UAV_4_DATA.NODEID=UAV_5_DATA.NODEID));

Query 2: target 5
SELECT   *
FROM     UAV_5_DATA, UAV_10_DATA
WHERE    ((UAV_10_DATA.LAT=609)) AND ((UAV_10_DATA.OBJPTR<=246)) AND ((UAV_5_DATA.OBJPTR=UAV_10_DATA.OBJPTR));

Query 3: target 9
SELECT   *
FROM     UAV_9_DATA, UAV_10_DATA, UAV_5_DATA
WHERE    ((UAV_9_DATA.LONG>351)) AND ((UAV_9_DATA.LAT>=40)) AND ((UAV_5_DATA.LONG<=804))
         AND ((UAV_9_DATA.OBJPTR=UAV_10_DATA.OBJPTR)) AND ((UAV_10_DATA.LAT= UAV_5_DATA.LAT));

Query 4: target 6
SELECT   *
FROM     UAV_6_DATA, UAV_10_DATA, UAV_4_DATA
WHERE    ((UAV_6_DATA.LAT<55)) AND ((UAV_6_DATA.NODEID<=260)) AND ((UAV_4_DATA.NODEID=22))
         AND (((UAV_6_DATA.TIMESTAMP=UAV_10_DATA.TIMESTAMP)) AND (UAV_10_DATA.OBJPTR= UAV_4_DATA.OBJPTR));

Query 5: target 9
SELECT   *
FROM     UAV_9_DATA, UAV_3_DATA, UAV_2_DATA, UAV_4_DATA WHERE ((UAV_9_DATA.TIMESTAMP<=764))
         AND ((UAV_9_DATA.LONG<102)) AND ((UAV_2_DATA.NODEID=66)) AND ((UAV_2_DATA.LONG>=614))
         AND ((UAV_9_DATA.LAT=UAV_3_DATA.LAT)) AND ((UAV_3_DATA.LAT=UAV_2_DATA.LAT))
         AND ((UAV_2_DATA.OBJPTR=UAV_4_DATA.OBJPTR));
```

Figure 4: Sample Queries Used

Method	Network 1	Network 2	Network 3	Network 4	Network 5	Network 6
Optimal Join		8.07				
Optimal semijoin	4.53	4.79	3.67	3.18	3.17	4.01
Top-K Cumulative Join	44.01	20.44	20.44	17.78	13.52	16.05
Top-K Cumulative semijoin	4.51	272.38	272.38	4.61	4.31	272.38
Top-K Iterative Join						
Top-K Iterative semijoin	5.18	5.55	14.48	3.15	4.17	4.47
Top-K Join-type Join	33.31	20.44	16.73	17.78	16.23	14.55
Top-K Join-type semijoin	6.81	4.55	4.47	4.81	4.17	6.06

Table 10: Heuristics V/S Optimal: Costs incurred across different connectivity configurations

Context Aware Ontology based Information Extraction

Sapan Shah and Sreedhar Reddy

Tata Research Development and Design Center,
Tata Consultancy Services Limited,
Pune 411013
India
{sapan.hs, sreedhar.reddy}@tcs.com

Abstract

We have developed an ontology based information extraction system where property and relation name occurrences are used to identify domain entities using patterns written in terms of dependency relations. Our key intuition is that, with respect to a given ontology, properties and relations are much easier to identify than entities, as the former generally occur in a limited number of terminological variations. Once identified, properties and relations provide cues to identify related entities. To achieve this, we have developed a pattern language which uses the grammatical relations of dependency parsing as well as linguistic features over text fragments. Ontology constructs such as classes, properties and relations are integral to pattern specification and provide a means for extracting entities and property values. The pattern matcher uses the patterns to construct an object graph from a text document. The object graph comprises entity, property and relation nodes. We have developed a global context aware algorithm to determine the ontological types of these nodes. Type of one node can help determine the types of other related nodes. We use the concept of entropy to measure the uncertainty associated with the type of a node. The type information is then propagated through the graph from low entropy nodes to high entropy nodes in an iterative fashion. We show how the global propagation algorithm does better than a local algorithm in determining the types of nodes. The main contributions of this paper are: an ontology aware pattern language; a global context aware type identification algorithm.

1. Introduction

We live in a networked world where information is growing at an explosive rate. The ability to draw useful insights from this information is going to be a key competitive advantage for enterprises. New business models are emerging that require highly dynamic configurations of supply chains. Effective management of such supply chains requires constant monitoring and analysis of information on suppliers, consumers, competitors, their operating environments and so on. This calls for a highly flexible and dynamic information architecture that allows us to collect and integrate information not only from within the enterprise but also from outside the enterprise such as online sources, social media sites and so on. The ability to dynamically discover and integrate relevant information sources is a key feature of this architecture.

With this in mind, we have developed an information integration architecture (see fig. 1) where ontologies and ontology driven information extraction play a key role. We have an enterprise level ontology that provides a unified view of information at the enterprise level. This ontology is mapped to source level ontologies. A source level ontology provides a conceptual view of information available at the source.

Integration of a new source into the framework involves specifying the relevant ontology and building an adaptor. The adaptor is responsible for extracting information and presenting it as an instance of the source ontology. Integration of structured sources is relatively easier and we will not discuss that in this paper. Integration of unstructured sources is more complex. First we have to identify the relevant ontology fragment (using ontology discovery techniques) and then we have to build a suitable information extraction component. Building an

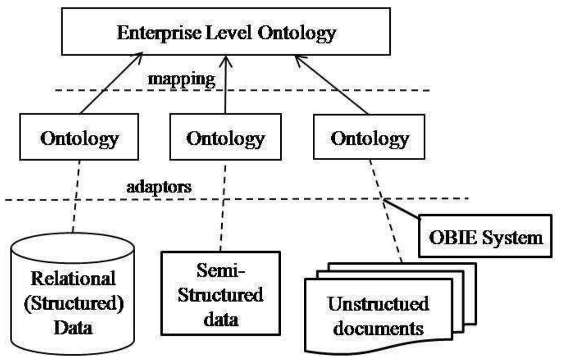


Figure 1: Enterprise Information Integration Framework

information extraction component using traditional IE techniques is a fairly involved job as they require extensive customizations (training, mark-up, tweaking rules, and so on). This is not a viable approach in a dynamic discovery and integration scenario. We need a more nimble approach. We discuss one such approach where information extraction can be driven entirely by the ontology, without any domain specific customizations. This obviously has its trade-offs. The approach places a higher premium on precision than on recall, as reliability of information is much more critical in a dynamic integration scenario where there is minimal expert intervention.

1.1. Ontology based Information Extraction

Information Extraction (IE) is the task of extracting structured information from unstructured or semi-structured sources. IE systems are supplied with the information of *what is to be extracted* in the form of output templates. Ontology based information extraction (OBIE) has recently emerged as a sub-field of IE where ontologies are used in the information extraction process. Output of the extraction process may also be represented in terms of an ontology. Ontology is defined as a formal and explicit specification of a shared conceptualization [11]. An ontology models a domain terminology in terms of concepts, properties and relations which can be used to specify information extraction targets. OBIE systems are broadly classified as ontology learning systems and ontology population systems. The task of an ontology learning OBIE system is to construct domain specific concepts and properties from unstructured text. Whereas, an ontology population OBIE system extracts instances of domain specific concepts and their property values for a given ontology. In this paper, our focus is on an ontology population system.

1.2. Our Approach

The key idea behind our approach is that it is much easier to identify property (and relation) name occurrences than entity name occurrences. The reason for this is that while an entity name may occur without an associated concept name reference, a property value rarely ever occurs without the associated property name reference. To illustrate, suppose we have an ontology fragment having one concept i.e. *Country* and two properties i.e. *Country.population* and *Country.capital*. Sentences such as the following are quite common:

`India has a population of 1.2 billion. Its capital is Delhi.`

While references to India frequently occur without the associated concept name reference (i.e. *Country*), it is difficult to imagine property values '1.2 billion' and 'Delhi' without the associated property name references (*population* and *capital*). Similarly it is difficult to imagine relation values without the associated relation name references. Also, while there can potentially be an infinite number of entity name occurrences, property (relation) names typically only occur in a limited number of terminological variations (e.g. population, populace). Thus in our approach we start by identifying occurrences of property and relation names and use them to identify entities. To achieve this, we have developed a pattern language which uses the grammatical relations (such as subject, verb, object, etc.) of dependency parsing to locate entities once the properties and relations are identified. The language also provides constructs to refer to ontology elements. These constructs serve two purposes: one, to specify constraints over ontology elements, and two to provide semantics for extracting information.

The pattern matcher uses the patterns written in the pattern language to construct an object graph from the input text document. The nodes of the object graph represent entities, properties and relations found in the document. The next step is to determine their ontological types for which we have developed a global context aware algorithm. We use the concept of entropy to measure the uncertainty associated with the type of a node. The type information is then propagated through the graph from low entropy nodes to high entropy nodes in an iterative fashion. The intuition behind this approach is that a node with a higher degree of certainty about its type can help determine the types of related nodes that have a lower degree of certainty about their types. For example, consider an ontology with classes such as *City*, *State* and *Country* and object property[1] *located_in* between: City and State; State and Country. Let's say a text document contains a sentence: `Gujarat is located in India`. Here the relation occurrence *located in* is not enough to decide the type of Gujarat which can potentially be *City* or *State*. Similarly, the type of India can be *State* or *Country*. Let's say the same document contains another sentence: `India is a country in South Asia`. This sentence provides the information that the type of India is *Country*. Now, if the information

[1] We will use the terms *relation* and *object property* interchangeably (similarly, *property* and *data type property*).

from *India* (a node with higher certainty about its type) is propagated to *Gujarat* (node with lower certainty), we can decide that the type of Gujarat is *State*.

The rest of the paper is organized as follows. Section 2 describes some of the systems developed for OBIE in the past. Section 3 discusses how domain ontology can be enriched to facilitate IE. Section 4 discusses details of input text pre-processing. Section 5 discusses pattern language constructs and their semantics. We present the global context aware type identification algorithm in section 6. Section 7 discusses experimental results. Section 8 ends with concluding remarks.

2. Related Work

Ontology based information extraction has recently emerged as a sub-field of IE. Research in this field has mostly concentrated on finding instances of domain specific concepts and learning taxonomic relations. Not much work has been done on finding non-taxonomic relations.

One of the first IE systems using ontology was the Embley's system [8] based on extraction ontologies, where ontologies are extended with regular expression based linguistic rules for ontological classes and properties. Other notable systems based on linguistic rules include FASTUS [1], PANKOW [4,5], OntoX [19], Ontosyphon [13], KIM [15]. FASTUS uses a cascade of finite state automata to extract the events and entities of interest. To extract instances of domain specific concepts, PANKOW, Ontosyphon and KnowItAll [9] systems use a set of generic Hearst patterns. These patterns are instantiated with ontological constructs for extraction purposes. For example, <Concept>s such as <Instance> is one of the Hearst patterns [12]. Here, Concept can be instantiated with country class to extract country instances. PANKOW [4,5] system first finds all proper nouns in a document and then conducts web based searches for every combination of proper noun and ontological class for a set of Hearst patterns. It then uses the number of hits recorded for each class to determine the correct class label for the proper nouns. Ontosyphon system uses a similar approach where instead of focusing documents, it uses web based searches to find possible instances of classes in the ontology. In addition to the linguistic rules, systems such as KIM [15] and iDocument [3] use gazetteer lists for some classes to facilitate IE.

As the constituency based parsers are typically closer to the syntactic structure than the semantics of the sentence, other parsing mechanisms such as dependency parsing, link grammar, etc. are used by different systems for relation extraction. Fundel et al. [10] have built RelEx system for BioInformatics domain. It uses dependency tree paths to extract interaction between genes and proteins. It uses gazetteer lists for extracting genes and protein names from natural language sentences. Similarly, Schutz and Buitelaar have developed RelExt system [17], where the goal is to extract relations between concepts for ontology learning. The authors motivate the use of verbs to express relation between classes that specify domain and range of some action or event. Similarly, Banko et al. [2] present open information extraction approach, where binary relationships between the entities can be obtained using verb-centric lexico-syntactic patterns.

IE systems perform linguistic processing (e.g. tokenizing, POS tagging, chunking, etc.) over the input text before the actual task of extraction. The generated linguistic features then can be used as part of extraction rules. Various NLP tools such as GATE[2], Stanford CoreNLP are used for this purpose. As we are building a new pattern language, it is worthwhile to compare it with JAPE[3] component of GATE. JAPE provides finite state transduction over document annotations based on regular expressions. It is used by Saggion et al. [16] and KIM [15] to write regular expression for entity extraction. It is possible to write ontology aware JAPE transducers where classes in the ontology can be referred as part of regular expressions. However, The JAPE regular expressions are written in terms of annotations over tokens while the pattern language we have developed can be used to write regular expression over trees in additions to tokens. Second notable difference is that, one has to write explicit JAVA code using GATE ontology APIs to store extracted information into the ontology. In our case, we have extended the pattern language itself with a set of constructs that specify how to store the extracted information into the ontology.

3. Ontology Enrichment for IE

To facilitate IE, ontologies in [8,19] are enriched with annotations. On similar lines, we have added following annotations to the domain ontology (A domain expert assigns values for these annotations).

- **Description**: The classes of ontology should be enriched with description annotations describing their meaning. This can be useful for assigning initial probability of an entity having a particular class type. For each class, the similarity between the words in the context of a given entity and the words in the class description is calculated. These similarity values are then normalized to get initial probability values.
- **Identification Weight**: For each ontological class, relative identification weights are assigned for its data and object properties. These weights indicate the relative importance of a property or relation in identifying the class. For example, consider an

[2] GATE – General Architecture for Text Engineering: java suite of tools to perform NLP tasks developed at University of Sheffield. (http://gate.ac.uk/)

[3] JAPE – Java Annotation Patterns Engine: regular expression language in GATE.

Organization domain with two classes i.e. *Employee*, *Department* and three properties i.e. *Employee.name*, *Department.name*, *Employee.reports_to*. Here, the occurrence of *reports_to* in text can provide cues that the type of the associated entity is *Employee*. The same is not true for *name*. Hence, *reports_to* is given more identification weight than *name*.

- **Synonyms**: While finding out property and relation mentions in the text, we also want to consider their synonyms. Hence, we provide an annotation to manually add synonyms for properties, relations and classes. The WordNet[4] synonyms can also be added as part of this annotation.
- **Value Patterns**: Stricter constraints on the values of data type property may be required in some cases. Hence, we enrich the ontology with value patterns annotation that specifies the regex patterns that the values of the data type property should match. For example, consider a Camera Review domain with a property *Camera.megapixel*. As observed in the Camera Review corpus, the regex for the value pattern can be: **\d+(\.\d+)?(mp|megapixel).**

The pattern matcher uses above mentioned annotation for identification as well as classification of entities and their property values.

4. Pre-Processing

IE from unstructured text generally employs a series of pre-processing steps where linguistic features of the input text are collected. This section describes these pre-processing steps and builds a data structure which will be used by the pattern matcher.

4.1. Linguistic features using Stanford CoreNLP

Stanford CoreNLP[5] is an integrated suite of natural language processing tools for English. The input text is first tokenized and passed to sentence splitter which converts the input text document into a sequence of sentences. The sentences are then POS tagged using Maximum Entropy based tagger. It uses Penn Tree bank tag set for POS tagging. The sentences are then parsed using lexicalized PCFG parser and the constituent parses are stored in a data structure. Stanford has also developed rules for converting phrase structure trees to dependency trees. A dependency tree provides a representation for grammatical relations between words in a sentence. It uses the concepts of dependency parsing [14] such as *relation*, *governor* and *dependent*. Pronominal co-reference resolution is also important for information extraction. We use Stanford's co-reference resolution system for this purpose.

4.2. Induced Tree data structure

As mentioned earlier, we use the grammatical relations of dependency parsing to locate entities once the property (relation) occurrences are found. We use Stanford dependencies for this purpose. Stanford dependencies (SD) [7] provide a representation of grammatical relations between words in a sentence. These relations are binary in nature and can be represented in the form of triplets: <name of relation, governor, dependent>. Few examples of the relations follow.

- **Nominal Subject** (*nsubj*): It is a noun phrase (dependent) which is a syntactic subject of a clause (governor).
- **Direct Object** (*dobj*): The direct object of a VP is the noun phrase (dependent) which is the (accusative) object of the verb (governor).

SD representation contains a total of 53 grammatical relations [6]. The words of a sentence along with their grammatical relations form a tree called dependency tree where nodes represent the words and edges represent the grammatical relations (as an example see figure 2a).

We need regular expressions to be matched over this tree structure as part of our pattern matching algorithm. Stanford provides Tree Regular Expression (TRegex): a utility for matching patterns in trees. The regular expressions in TRegex are written in terms of node labels and they do not consider edge labels. The regular expression patterns that we need to apply use edge labels in addition to node labels. To solve this problem, we created a tree data structure different but derived from

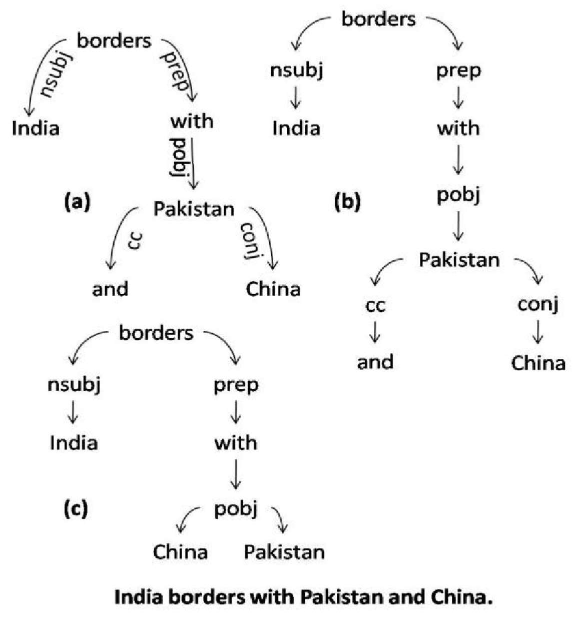


Figure 2: Induced Tree data Structure for an Example Sentence

[4] WordNet is a large lexical database of English developed at Princeton University. (http://wordnet.princeton.edu/)

[5] Stanford CoreNLP: a set of natural language analysis tools provided by Stanford University. (http://nlp.stanford.edu/software/corenlp.shtml)

Table 1: Tree Transformation Patterns

TreeTransformation	TRegex Pattern - condition	TSurgeon Operations	Remarks
ConjunctionAnd	/.*/=head < (cc=vCC < and=vAnd) < (conj=vConj < /.*/=brother)	move brother $- head; delete vConj	All the conjuncts in *and* conjunction becomes siblings; children of Parent of head conjunct. (India borders with *Pakistan **and** China*)
CompoundNoun	/.*/=head < (nn=vNN < /.*/=compound)	accumulate compound head compound; excise vNN compound	Words in a compound noun are considered as single unit e.g. India borders with Sri Lanka; *Sri Lanka* is stored as a single induced tree node.
ModifierList	/.*/=head < (/.*mod.*/=vMod < /.*/=modifier)	accumulate modifier head modifier; excise vMod modifier	All modifiers are stored along with an induced tree node of word that they modify.
CompoundNumber	/.*/=head < (number=vNumber < /.*/=compound)	prune vNumber	All the words in compound number are treated as a single node *e.g.* I lost $ 3.2 billion. Here, *$ 3.2 billion* is treated as a single node of number type.

dependency tree. This data structure contains nodes for words as well as grammatical relations as shown in figure 2b. The grammatical relation nodes are internal nodes: used only for patterns, not for extraction. We will refer to this data structure as *induced tree* in the rest of the paper. It should be noted that Stanford provides a utility for pattern matching over dependency trees called Semgrex. However, it does not provide any means of integrating ontology information.

Node description in a TRegex pattern is specified using literal or regular expression (specified between /). During pattern matching, it matches with node labels of the tree. Relations are specified between the node descriptions. All relations in a pattern are relative to the first node. Parenthesis can be used to group related nodes. For example, A < B < C mean A is the parent of B and C; A < (B < C) means A is a parent of B and B is a parent of C. Named nodes are used to bind a variable with the value matching the specified regex. For example, /NN.*/=Var is a named node and variable Var can be used to refer to the actual node label that matches with regex NN.*.

4.3. Tree Transformations

A set of tree transformations are applied to the induced tree before the actual pattern matching starts. Stanford provides TSurgeon - a tree transformation language. TSurgeon pattern consists of a single TRegex pattern P and a number of TSurgeon operations that are executed when P matches on the tree. These operations refer to the named nodes in the TRegex pattern for tree manipulations. Suppose we want to perform IE for GeoPolitical Entities domain having a *Country* class and *borders_with* relation. Figure 2b shows an induced tree for a sentence from this domain. A TRegex pattern to extract this relation is

$$/.*/=\text{Verb} < (\text{nsubj} < /.*/=\text{Source}) < (prep < (with < (pobj < /.*/= \text{Target}))) \quad (\mathbf{1})$$

Where, Verb, Source and Target are TRegex variables. When this pattern is applied to the induced tree, it returns a match where the variable bindings for Verb, Source and Target are *borders*, *India* and *Pakistan* respectively. As Verb matches with the relation name, we can extract an RDF triple viz. (India, borders_with, Pakistan). If we observe the example sentence closely, we missed extracting one more RDF triple viz. (India, borders_with, China). To solve this problem, the induced tree needs to be transformed such that China-Node becomes the child of pobj-Node. Stanford dependencies handle *and* conjunctions the following way: one of the conjuncts is selected as head (Pakistan here); the rest of the conjuncts become children of the head conjunct with conjunction (*conj*) relation. Let's denote the parent of the head conjunct as H (pobj here). First we need to apply a TRegex pattern to find *and* conjunction and then apply TSurgeon operations such that all the conjuncts become children of H. Figure 2c shows the induced tree after the application of this tree transformation (see table 1: ConjunctionAndTransformation). As we can see, the missed RDF triple can be extracted now, as it matches with the TRegex pattern in 1. Table 1 lists a set of tree transformations we have used.

5. Pattern Language

We have developed a pattern language for processing the induced tree and extracting information. Due to space constraints, we present only a subset of the grammar of this language (see text box 1 below).

A pattern consists of a premise and a sequence of

```
patterns:- pattern* <EOF>
pattern:- patternID "{" premise "}"
          "->" "{" actions "}"
patternID:- (DIGIT)+
premise:- (treePath ";")+
       (ontologyConstraint ";")+
              ("{" boolean_expression
               "}" ";")?
treePath:-element| element"--" treePath
ontologyConstraint:-
       ontologyElement = variable
actions:- ("{" action + "}")+
action :- LHS = RHS ";"
LHS:- ontologyActionElement | variable
RHS:-variable |identifier
       |action_function
```

1. Grammar for Pattern Language

actions. A premise is a set of conditions that should hold true for the actions to be executed. It consists of,

- **Tree paths**: A tree path specifies a sequence of elements. These elements are matched against node labels in the induced tree. An element can be a variable, identifier or a regular expression. A variable can be bound or unbound. While an unbound variable is bound with a value during pattern matching, a bound variable specifies a constraint: a matching tree node label must have the same value.
- **Ontology Constraints**: An ontology constraint is of the form '<lhs> = <rhs>'. It specifies that the value bound to a variable on the right hand side (rhs) must match with an ontology element on the left hand side (lhs). An ontology element can be a class, property, relation or an instance. Looking at the example sentence in figure 2, one would like to check whether the variable Verb gets a binding that matches with some ontology relation or its synonyms (which happens to be *borders_with* in the example). If so, we have a possible relation extraction with corresponding source and target entities. We can specify this constraint using

  $$\boldsymbol{relation = < Verb >}$$

  This way, our pattern language provides language constructs to explicitly refer to various ontological elements.
- **Boolean Expression**: We support two boolean operators: And, Or. The basic operand in a boolean expression is a Boolean function. We support boolean functions over ontological constructs as well as linguistic features. For example, to check whether the type of the value bound to a variable matches with a pre-defined data type in the ontology, we have a function – isTypeMatching.

The actions component in the pattern specifies a sequence of actions to be performed over variable bindings from the premise. The basic constituent used in an action is assignment. An assignment is of the form '<lhs> = <rhs>'. The left hand side (lhs) of an assignment can either be a variable or an ontology element (ontologyActionElement in the grammar). We have a set of predefined keywords to refer to ontology elements with the following semantics,

- **relation (property)**: value of the right hand side (rhs) expression must be interpreted as an object (data) property in the ontology.
- **class**: value of the rhs expression must be interpreted as a class (concept) in the ontology.
- **source (target)**: value of the rhs expression must be interpreted as a source (target) entity of the property or relation occurring in the action.
- **entity**: value of the rhs expression must be interpreted as an entity (class instance) in the document.
- **previous_entity**: value of the rhs expression must be interpreted as an entity matched in the previous sentence in the document.

When lhs is a variable, it specifies that the values of both lhs and rhs expressions refer to the same underlying domain entity. This essentially says that lhs and rhs are to be treated as aliases of the same domain entity.

The rhs expression of an assignment can be,

- **Variable**: bound value of the variable is used in the action assignment.
- **Literal**: literal value specified as an identifier is used in the action assignment.
- **Action Function**: We may want to manipulate the bound value of a variable before it can be used for extraction. To do this, we provide action functions. The value returned by executing the action function is used as an action assignment. For example, if we have an instance of country and want to assign value for the official name of the country, we can use a function – concat(Republic, of, <Country>). During execution, if variable Country is bound to India, we can get the official name Republic of India using this function.

As mentioned, an action is specified by a group of assignments. For example, a relation extraction with source and target entities are specified by,

{source=<Entity1>;target=<Entity2>;relation=<Relation>}

Similarly there are actions to specify extraction of property with source entity and target value; extraction of class instance pair; extraction of an equivalent name (name aliases) for an entity (India and Republic of India).

5.1. Example Patterns

We will go through an example to see how one specifies patterns in this language. Consider GeoPolitical Entities domain with a *Country* Class and *coastline* property. Let's look at a sample sentence (Table2 – Pattern 1):

```
India has a coastline of 7517 km.
```

In the dependency tree of this sentence, `has` is the root verb; `India` is a subject and `coastline` is a direct object of `has`; `7517 km` is the prepositional object of preposition-`of` which modifies the direct object. So, paths that a pattern should look for in the induced tree are,

has -- dobj -- <Property> -- prep -- of – pobj -- <Value>;
has -- nsubj -- <Entity>

In addition, the direct object should match with some data type property in the ontology. An ontology constraint to specify this would be,

property=<Property>;

The premise built using paths and an ontology constraint above can match any data type property in the ontology hence it matches with `coastline`. The actions part for this pattern should perform property extraction and can be specified as,

{source=<Entity>;target=<Value>;property=<Property>}

Table 2: Generic Domain Independent patterns - Examples

India has a coastline of 7515 km.	**property extraction**

1 {
<HAS=has> -- dobj -- <Property> -- prep -- of -- pobj -- <Value>; property = <Property>;
<HAS> -- nsubj -- <Entity>; {isRoot(<HAS>) && isTypeMatching(<Value>, Number)};
} -> {
source=<Entity>; target=<Value>; property=<Property> }	

Ratan Tata launched Tata Nano in 2010.	**relation extraction**

2 {
<Verb> -- nsubj -- <Subject>; <Verb> -- dobj -- <Object>;
relation = <Verb>; {isRoot(<Verb>)};
} -> {
source = <Subject>; target = <Object>; relation = <Verb>; }	

India is a country in South Asia.	**Class Identification**

3 {
<Concept> -- nsubj -- <Instance>; <Concept> -- cop;
class = <Concept>;
} -> {
class = <Concept>; entity = <Instance>; }	

If we look closely at the dependencies exhibited in this example sentence, they are generic and can happen across sentences from different domains. As long as a sentence has a direct object matching with a data type property from a domain specific ontology, the entity and property value extraction is possible. In that sense the pattern described above is generic and can be used across different domains. We have compiled a set of such generic, domain independent patterns. There are a total of 18 patterns out of which we list only 3 patterns in Table 2 due to space constraints. First two patterns in the table are based on property extraction and relation extraction respectively. The last pattern shows one of the class identification patterns.

6. A Greedy Algorithm for Type Identification

We will first describe the ontology we have used for our experiments. We will be referring to it in the rest of the paper. We have downloaded FAO (Food and Agriculture Organization of the United Nations) Geopolitical ontology and modified it for our experiments. Figure 3 shows a section of this ontology.

As described in section 4 and 5, the text document is

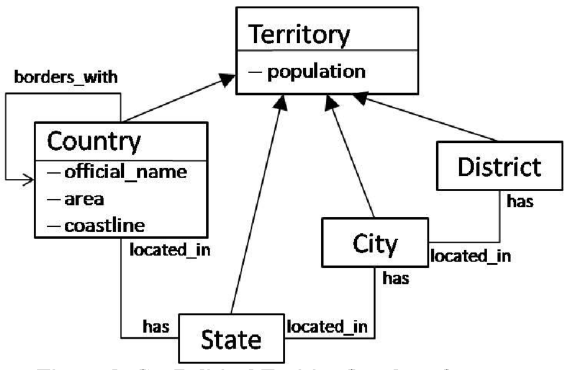


Figure 3: GeoPolitical Entities Ontology fragment

converted to a sequence of induced trees. The pattern matching algorithm then applies a set of patterns on these trees and generates a graph structure. We will refer to this graph structure as *object graph* in the rest of the paper. The object graph contains three types of nodes *viz.*

- **Entity Node**: represents an instance of a domain entity found in the document.
- **Property Node**: links an entity node with its property values.
- **Relation Node**: links two entity nodes that represent domain and range of some ontological object property.

These nodes just represent the entities, properties and relations identified in the document; their ontological types still have to be determined. The possible ontological types for the three types of nodes are: classes for entity nodes; data properties for property nodes; object properties for relation nodes. As we also account for co-references, the same entity node is used if the entity is referred in different parts of a document.

Table 3 gives a simple algorithm to determine the ontological types for the nodes in the object graph. We will refer to this algorithm as *LocalIE* in the rest of the paper. The first step in the algorithm applies a set of class-identification patterns to determine types for the entity nodes. We have used the Hearst patterns [10] for class-identification. The type of an entity which matches these patterns can directly be inferred; one does not have to rely on property or relation occurrence for its type identification. For example, consider a sentence: `India is a country in South Asia.` The type for the entity *India* can directly be determined using the pattern: <*Instance*> is a <*Concept*>. Pattern 3 in table 2 captures this pattern in terms of dependency relations. For the entity nodes which do not match these patterns and for the property and relation nodes, the algorithm assigns equal scores for their ontological types.

Table 3: An Algorithm for IE using Local Context

LocalIE – An Algorithm for IE using local context

1. Apply class-identification patterns (e.g. Table 2-Pattern 3) to get the type information for the entity nodes in object graph.
2. Use formula 2 to find the types of property nodes (Similarly find the types of relation nodes).
3. For each entity node (whose type is not determined in step 1):
   - **a.** Find the score for each ontology class using formula 3. As shown, this formula uses the local context (related property and relation nodes) along with their identification weights.
   - **b.** Assign class with the highest score as the correct type for the entity node (formula 4).
4. Convert the object graph to RDF triples.

$$type(A) = \underset{1\le i\le |property|}{\operatorname{argmax}} \{similarity(P_i.words, A.words)\} \tag{2}$$

where,
P_i= ith data property in the ontology;
P_i.words = words in the data property P_i (including its synonyms);
A.words = words occurring in the property node A.

$$score\left({C_i}/{E}\right) = \sum_{j=1}^{|property|} \left[score\left({P_j}/{A}\right) * C_i.iWeight(P_j)\right] + \sum_{k=1}^{|reltion|} \left[score\left({R_k}/{B}\right) * \sum_{L\in Range(R_k)} [C_i.iWeight(R_k L)]\right] \tag{3}$$

where,
E is an entity node in focus having a property node A and a relation node B.
$C.iWeight(P)$ = identification weight of property P for class C;
$C.iWeight(RL)$ = identification weight of relation R for class C; $L \in Range(R)$;
C_i= ith class in the ontology; P_j= jth property in the ontology;
R_k= kth relation in the ontology.
$score(T/N)$ = score for an ontological type T given node N.

$$type(E) = \underset{1\le i\le |class|}{\operatorname{argmax}} \left\{score\left({C_i}/{E}\right)\right\} \tag{4}$$

The types for the property and relation nodes are found by matching them with ontological data and object properties respectively (step 2). Here, the edit-distance based similarity scores are calculated between the words of a property (relation) node and an ontology data (object) property. The synonyms of a data (object) property are also taken into account. The data (object) property with the highest similarity score is then chosen as the correct type for the property (relation) node (formula 2). To determine the type of an entity node, the scores found for the neighboring property and relation nodes as well as their identification weights are used (formula 3). This algorithm uses only the local context to find the correct type of an entity node.

More informed decision for the type of an entity node can be made if the global context is also taken into account. Let us first motivate the need of such a global context aware algorithm. Consider the ontology in figure 3. It contains an object property *located_in* between *State* and *Country*; *City* and *State*; *District* and *City*. Whenever this relation occurs in the text document, there is an ambiguity about the types of the source and target entity nodes as the same name is used to refer to three different object properties in the ontology. Consider a text fragment from this domain,

```
Surat is located in Gujarat. It is recognized for its textile and diamond businesses. Vadodara is also located in Gujarat. It is the third most populated city with a population of almost 1.6 million.
```

The underlined phrases in this fragment are the instances of domain entities and their properties (relations). As we can see in the first sentence, the relation *located in* cannot provide correct type information for the related entities i.e. *Surat* and *Gujarat*, as they may refer to any of the four classes *viz.* District, City, State or Country. However from the last sentence, we can easily infer that the type of the entity *Vadodara* is City. If we use the type information of *Vadodara* along with the *located in* relation in the third sentence, we can infer that the type of *Gujarat* is State. Now, if we use the type information of *Gujarat* in sentence 1, we can infer that the type of *Surat* is City. The local algorithm we described in table 3 neither takes global context into account nor performs this kind of information propagation.

6.1. Entropy - Information Theory

We use the concept of entropy from information theory [18] to quantify the uncertainty associated with the type of a node. Entropy is a measure of uncertainty associated with a random variable and defined in terms of its probability distribution. Let's denote X as a discrete random variable having a set of possible values $\{x_1, x_2, \dots, x_n\}$ and a probability mass function $p(X)$ (such that $\forall i: p(x_i) \in [0,1]$; $\sum_{i=1}^{n} p(x_i) = 1$;). The entropy of X is then defined as,

$$H(X) = -\sum_{i=1}^{n} p(x_i) * \log_b p(x_i) \tag{5}$$

For example, consider two experiments: tossing a fair coin ($p(head) = 0.5$ and $p(tail) = 0.5$); tossing a two-headed coin ($p(head) = 1$ and $p(tail) = 0$). The outcome of the former experiment is most uncertain and thus has highest entropy, while the later has a definite

Table 4: An Entropy based Greedy Algorithm for IE

```
GlobalIE – An Entropy based Greedy Algorithm for IE

1. Execute step 1 to step 3a of the LocalIE algorithm to
   determine the types of property and relation nodes, and to
   get initial scores for entity nodes.
2. Normalize the class-score for each entity node $E$ such
   that,
   $\forall i: 1 \le i \le |class|; score(C_i/E) \in [0,1]; \sum_{i=1}^{|class|} score(C_i/E) = 1$
   Calculate entropy values of all entity nodes.
3. Create a min-priority queue $Q$; add all entity nodes in $Q$.
     visited_nodes = $\phi$;
4. While($Q$ != empty) {
     $E$ = remove a node from $Q$ with the least entropy value;
     Assign correct type for node E using formula 4.
     Add E to visited_nodes;
     **propagate_score(visited_nodes, E);**
     }
5. Convert the object graph to RDF triples.

**propagate_score(visited_nodes, entity_node E)** {
    For(each relation $B$ where $E$ is the source entity) {
        $X$ = target entity for relation $B$;
        propagetIfLow(E, X);
    }
    For(each relation $B$ where $E$ is the target entity) {
        $Y$ = source entity for relation $B$;
        propagateIfLow(E, Y);
    }
}
**propagateIfLow(entity_node E, entity_node A)** {
    If $(entropy(A) > entropy(E)\ AND$
                 $A \notin visited\_nodes)$) {
        For each class $C_i; 1 \le i \le |class|$,
            Update $score(C_i/A)$ using formula 6.
        Normalize the class-score for node $A$;
        Re-calculate the entropy of node $A$ ;
        **propagate_score(visited_nodes, A);**
    }
}
```

$$score\left({}^{C_i}/_{D}\right) += \sum_{j=1}^{|relation|} \left[score\left({}^{R_j}/_{B}\right) * \left(\sum_{L \epsilon Range(R_j)} C_i.iWeight\left(R_j L\right) * score\left({}^{L}/_{E}\right) \right) \right] \quad \textbf{(6)}$$

where, D and E are source and target of relation node B

outcome and the entropy is 0. The entropy of a random variable is proportional to the uncertainty of the outcome.

In our context, we use the concept of entropy to measure the uncertainty associated with the type of a node. For example, for an entity node the possible types are the classes in the ontology. Let $C_i; 1 \le i \le |class|$ denote the classes. If we do not have any information about the type of an entity node E (highest uncertainty and entropy), we assign uniform score for the classes i.e. $score(C_i/E) = 1/|class|$. In our algorithm, we use the local formula in 2 to assign initial scores for the class types of the entity nodes.

6.2. An Entropy based Greedy Algorithm

To find the correct type of an entity node, the LocalIE algorithm just uses the neighbouring property and relation nodes. If the information about the correct type of some entity node in the object graph is available, it should be used for classification of other related entity nodes in the graph. Table 4 describes a global context aware algorithm which uses related entity nodes in addition to the property and relation nodes for classification. We will refer to this algorithm as *GlobalIE* in the rest of the paper.

GlobalIE uses edit-distance based similarity score for classifying property and relation nodes (same as LocalIE). The main difference is the use of related entity nodes to classify current entity in focus. The entity nodes in the object graph are ordered according to their entropy values. The rationale behind this ordering is: the nodes with high information about their correct type can help determine the types of other related nodes having low information about their types.

In GlobalIE, once the types for the property and relation nodes are determined, the entity nodes are added to a min-priority queue (step 3). The nodes in this queue are ordered in the increasing order of their entropy values. To calculate the entropy value correctly, the scores for the class types of an entity node E must satisfy two conditions:$\forall\ i: 1 \le i \le |class|$; $score(C_i/E) \in [0,1]$ and $\sum_{i=1}^{|class|} score(C_i/E) = 1$. To achieve the same, we normalize these scores in the following way $\forall i: 1 \le i \le |class|$; $score(C_i) = score(C_i)/\sum_{k=1}^{|class|} score(C_k)$. During each pass of the while loop in step 4, an entity node with the least entropy value is removed from the queue and assigned its correct type using formula 4. The information contained in this node is then propagated to other nodes through the graph structure. In particular, the type information is propagated through the graph from low entropy nodes to high entropy nodes (see function: propogate_score). As we do not want to update the score of a node which is already assigned its type, we maintain a list of visited nodes (visited_nodes list in step 3). The time complexity of GlobalIE is in the order of the size of the object graph. Let's now go through an example to demonstrate how the information is propagated between the nodes and how the entropy based ordering is beneficial for entity classification.

6.3. An Example demonstrating Global IE

Consider again the GeoPolitical entities domain (fig. 3) and the example text fragment mentioned earlier in this section. We used a set of generic patterns as described in section 5 for information extraction and applied the pattern matcher over this fragment. Figure 4 shows the

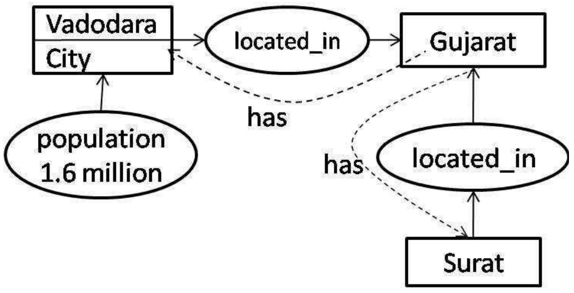


Figure 4: Object graph for text fragment

generated object graph. Let's now go through the execution of GlobalIE. Table 5 shows the scores of class types of the entity nodes and their entropy values at various points in time during the execution of the algorithm. Initially, the scores are equal for all entity nodes (except *Vadodara*, as it is directly assigned its correct type by the class-identification pattern) as shown in row 1. The scores of the property and relation nodes (along with their identification weights) are then used to update the scores of the entity nodes (step 1). Row 2 shows these scores after normalization (step 2). During the first pass of the while loop in step 4, *Vadodara* is selected and removed from the priority queue, as it has the least entropy value. The scores of the class types of *Vadodara* are then propagated through the graph structure. The object graph has a relation node *located_in* for which *Vadodara* is a source entity and *Gujarat* is a target entity. Hence, the scores for the class types of *Gujarat* are updated using the scores of *Vadodara* (row 4). In the second pass, *Gujarat* is selected and removed from the priority queue as it has the least entropy value now. The class type of this node is then determined using formula 4. Now, this node is connected to two entity nodes in the object graph i.e. *Vadodara* and *Surat*. As the entity node *Vadodara* is already visited earlier, it is ignored and the scores for the class types of *Surat* are updated using the scores of *Gujarat* (row 5). In the third pass, we are left with only one entity node i.e. *Surat*. Hence, it is selected and removed from the priority queue (row 6) and its class type is determined using formula 4. The priority queue is empty now and the algorithm terminates. The class types assigned by this algorithm for the entity nodes are *City*, *State* and *City* for *Vadodara*, *Gujarat* and *Surat* respectively. As we can see, the algorithm finds the correct values for the class types of the entity nodes. When we executed LocalIE algorithm on the same text fragment, it incorrectly assigned class types *District* and *Country* for the entity nodes *Surat* and *Gujarat* respectively (The class type having highest score in table 5 - row 2 is selected as the correct type of the entity node in LocalIE).

7. Experiments

7.1. Digital Camera Reviews domain

Yildiz et al. [19] have developed an ontology driven IEs – OntoX. It focuses mainly on identifying property mentions and their values. The ontology contains one class i.e. *camera* having five data properties. It is enhanced with a set of keywords for each data type property. The system uses regular expressions to find the instances of pre-defined XML data types in the text document and looks for keywords in their vicinity. The property whose keyword is closest to the data type instance and having the same XML data type is selected. For example, consider a sentence: Powershot A95 is a 5.0 megapixel camera. Here, 5.0 is XSD:float and megapixel is a property having keyword megapixel and data type XSD:float. Hence, 5.0 is a value of megapixel property. The dataset consists of 138 digital camera reviews. The focus of this experiment is to show how the patterns based on grammatical relations are useful for relating entities with their property values.

It should be noted here that the task performed by OntoX system is to just find property values. In our case, we also find entities and associate them with their property values. We have used the set of generic patterns described in section 5.1 for IE over camera reviews dataset. Table 6 shows the precision and recall values for some of the camera properties. The precision of our system is better than the OntoX system while the recall values are very low. The reason is in our approach we only identify those properties for which entities are identified. Thus, we miss some of the properties. Whereas OntoX focuses only on property values, so its recall is higher. It is interesting to note that we get very high precision values which suggest that our approach is conservative. The system may not be able to extract all the entities and property values but whatever is extracted

Table 5: The scores of class types of the entity nodes in the example text fragment. The first column specifies the algorithm step; the rest of the columns specify the scores of class types of the entity nodes using the format: (Territory, State, District, Country, City)

Step	Gujarat	Vadodara	Surat
Init.	(0.2, 0.2, 0.2, 0.2, 0.2) – 1.61	(0, 0, 0, 0, 1) - 0	(0.2, 0.2, 0.2, 0.2, 0.2) – 1.61
2	(0.05, 0.25, 0.05, 0.41, 0.25) – 1.35	(0, 0, 0, 0, 1) - 0	(0.08, 0.24, 0.37, 0.08, 0.24) – 1.44
While loop of step 4			
pass 1	(0.03, 0.48, 0.03, 0.28, 0.17) – 1. 24	**(0,0,0,0,1) - 0**	(0.08, 0.24, 0.37, 0.08, 0.24) – 1.44
pass 2	**(0.03, 0.48, 0.03, 0.28, 0.17) – 1. 24**	(0,0,0,0,1) - 0	(0.05, 0.17, 0.26, 0.05, 0.47) – 1.31
pass 3	(0.03, 0.48, 0.03, 0.28, 0.17) – 1. 24	(0,0,0,0,1) - 0	**(0.05, 0.17, 0.26, 0.05, 0.47) – 1.31**

Table 6: Comparision of Our System with OntoX on Camera Review domain

Property	Our System		OntoX	
	Prec.	Rec.	Prec.	Rec.
Megapixel	0.93	0.39	0.52	0.51
Display Size	0.88	0.2	0.80	0.82
Model Name	0.76	0.64	0.79	0.79

Table 7: Results on GeoPolitical Entities Domain

Concept/Property /Relation	Precision	Recall
Country	0.85	0.69
borders_with	0.72	0.39
located_in	0.86	0.78
official_name	1.0	0.74
population	0.92	0.57
coastline	0.57	0.80
area	1.0	0.60
Total	**0.82**	**0.54**

is extracted with high accuracy. If we look at the recall values closely, the recall for the property model_name is high. It then decreases for megapixel and very low for display_size. If we observe any file from the corpus, the model_name property is same as the name of an extracted entity. The megapixel property occurs very near to the entity occurrence (mostly in the same sentence). The display_size property is mentioned very far from the entity (mostly in the next paragraph), thus decreasing the probability of associating the property with the entity. The induced tree paths used in our patterns do not consider word relations across sentences. We rely on co-reference resolution when the entity and property mentions are in different sentences. We have also provided a language construct called previous_entity using which a pattern can refer to the entities found in earlier sentences. Despite this, it is not easy to relate an entity with its property if they are widely separated in the text.

7.2. GeoPolitical Entities domain

We have downloaded 36 Wikipedia pages of country profile, converted them to text and manually tagged them for correct entity and property values. As part of this experiment, we have considered the data and object properties of only the *country* class (see fig. 3). We used the generic patterns described in section 5.1 for IE. Our experiments helped us identify these patterns and during the course of the experiments our initial set went through several additions and modifications. We randomly selected 10% of corpora (4 pages) to analyze whether the generic patterns we have are good enough for extraction, especially we looked at the entity, property and relation occurrences and how they are related by the dependency relations. At the end of this exercise, we had to add 3 new patterns and modify 4 existing patterns. In total we used 14 patterns and performed the experiments. Table 7 lists the precision and recall values for classes, properties and relations. The overall precision is 0.82 and recall is 0.54 which again strengthens our argument that the system is conservative and makes fewer mistakes (high precision). The reason for higher precision is that unlike in traditional approaches where identification is primarily text pattern based (which can throw up spurious matches), we also consider an entity's property and relationship context which reduces spurious matches. However, this can have an adverse impact on recall as some of the valid matches might also be turned down on account of not having matching property and relation contexts. As explained earlier, this behaviour of higher precision and lower recall is fine, as reliability is a key concern in our enterprise information integration framework.

We would like to point out here that the extra patterns that we had to add were due to the peculiar ways in which some properties were written in the text corpora. The generic patterns we have collected will work best when the sentences in the text document are property formed and follow the English grammar, such as in published articles. The text documents in different genres may have different styles of writing English sentences (publications vs. blog posts) and it's important to capture them in the form of dependency relations. For this reason, we may have to analyze different genres of text documents and augment the list of generic patterns.

7.3. Analysis of our OBIE system

The key constituents of our system are: a pattern language and a global type identification algorithm. A relevant question in this context is what varieties of patterns can be expressed in our pattern language. The constituents of the language (dependency relations, boolean functions, ontology constraints) provide the necessary power to write various kinds of patterns mentioned in the IE literature. A lot of systems in the literature have used Hearst pattern [12] and lexico-syntactic patterns [2] for extraction. We could successfully convert these patterns into equivalent patterns in our pattern language.

Once the object graph is generated by the pattern matcher, the type of the object graph nodes has to be identified. The accuracy of type identification can improve if we go beyond the local context and make use of all the relevant information available in the document. That's what our global propagation algorithm aims to achieve. The direction of propagation is determined by entropy ordering where information flows from nodes of high certainty to nodes of low certainty. In many cases mere presence of properties and relations is sufficient to uniquely identify an entity's type. This is possible when the names of these properties and relations are unique in the ontology. However duplicate names are quite common in real-life ontologies. For example, the *located_in* object property given in section 6 relates three different class pairs. Similarly, *reports_to* structure in an organization ontology; *part_of* structure in a product ontology, and so on. A global propagation algorithm can make a big difference in such cases.

8. Conclusion and Future Work

We presented an information extraction approach where we first identify property and relation name occurrences in the text and then use patterns written in terms of dependency relations to identify related entities. To achieve the same, we have developed an ontology aware pattern matcher which uses these patterns to generate an object graph from a text document. We have also developed a global context aware algorithm to identify the ontological types of the object graph nodes. The algorithm is greedy and it uses the entropy ordering to decide information propagation between the nodes where type information is passed from low entropy nodes to high entropy nodes. The main contributions of this paper are: an ontology aware pattern language; a global context aware type identification algorithm.

We have experimented with GeoPolitical entities domain with a small set of text documents from Wikipedia. The result looks promising. An immediate (also important) task at hand is to test our approach on larger and varied set of corpora to check its applicability in general. We also want to integrate our system into the larger enterprise information integration framework to check its utility.

References

[1] Douglas E. Appelt, Jerry R. Hobbs, John Bear, David J. Israel, and Mabry Tyson, "FASTUS: A Finite-state Processor for Information Extraction from Real-world Text," in *IJCAI*, Chambéry, France, 1993, pp. 1172-1178.

[2] Michele Banko, Oren Etzioni, Stephen Soderland, and Daniel Weld, "Open information extraction from the web," *Communication of ACM*, vol. 51, no. 12, pp. 68-74, 2008.

[3] Adrian Benjamin, Hees Jorn, van Elst Ludger, and Dengel Andreas, "iDocument: Using Ontologies for Extracting and Annotating Information from Unstructured text," in *KI*, 2009, pp. 249-256.

[4] Philipp Cimiano, Siegfried Handschuh, and Steffen Staab, "Towards the self-annotating web," in *Proceedings of the 13th international conference on World Wide Web*, NY, USA, 2004, pp. 462-471.

[5] Philipp Cimiano, Gunter Ladwig, and Steffen Staab, "Gimme' the context: context-driven automatic semantic annotation with C-PANKOW," in *Proceedings of the 14th international conference on World Wide Web*, Chiba, Japan, 2005, pp. 332-341.

[6] Marie-Catherine de Marneffe and Christopher D. Manning, "Stanford typed dependencies manual," Stanford University, 2008.

[7] Marie-Catherine de Marneffe and Christopher D. Manning, "The Stanford typed dependencies representation," in *22nd International Conference on Computational Linguistics*, Manchester, United Kingdom, 2008, pp. 1-8.

[8] David W. Embley, "Towards Semantic Understanding -- An Approach Based on Information Extraction Ontologies," in *Proceedings of the Fifteenth Australasian Database Conference*, Dunedin, New Zealand, 2004, pp. 18-22.

[9] Oren Etzioni et al., "Web-scale information extraction in knowitall: (preliminary results)," in *Proceedings of the 13th international conference on World Wide Web*, New York, NY, USA, 2004, pp. 100-110.

[10] Katrin Fundel, Robert Kuffner, and Ralf Zimmer, "RelEx - Relation extraction using dependency parse trees," *Bioinformatics*, vol. 23, pp. 365-371, 2007.

[11] Thomas R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199-220, July 1993.

[12] Marti A. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *14th Internation Conference on Computational Linguistics*, Nantes, France, 1992, pp. 539-545.

[13] Luke K. McDowell and Michael Cafarella, "Ontology-driven information extraction with ontosyphon," in *ISWC*, Athens, GA, 2006, pp. 428-444.

[14] Joakim Nivre, "Dependency Grammar and Dependency Parsing," Vaxjo University: School of Mathematics and Systems Engineering, 2005.

[15] Borislav Popov, Atanas Kiryakov, Damyan Ognyanoff, Dimitar Manov, and Angel Kirilov, "KIM – a semantic platform for information extraction and retrieval," *Natural Language Engineering*, vol. 10, no. 3, pp. 375-92, 2004.

[16] Horacio Saggion, Adam Funk, Diana Maynard, and Kalina Bontcheva, "Ontology-Based Information Extraction for Business Intelligence," in *ISWC*, 2007, pp. 843-856.

[17] Alexander Schutz and Paul Buitelaar, "RelExt: A Tool for Relation Extraction from Text in Ontology Extension," in *ISWC 2005*, 2005.

[18] E. Claude Shannon, "A mathematical theory of communication," *Bell System technical journal*, vol. 27, pp. 379-423, 1948.

[19] Burcu Yildiz and Silvia Miksch, "ontoX - a method for ontology-driven information extraction," in *ICCSA'07*, vol. 3, Kuala Lumpur, Malaysia, 2007, pp. 660-673.

REBOM: Recovery of Blocks of Missing Values in Time Series

Mourad Khayati
Department of Informatics
University of Zürich
Binzmühlestrasse 14, CH-8050
Zürich, Switzerland
mkhayati@ifi.uzh.ch

Michael H. Böhlen
Department of Informatics
University of Zürich
Binzmühlestrasse 14, CH-8050
Zürich, Switzerland
boehlen@ifi.uzh.ch

ABSTRACT

The recovery of blocks of missing values in regular time series has been addressed by model-based techniques. Such techniques are not suitable to recover blocks of missing values in irregular time series and restore peaks and valley. We propose REBOM (REcovery of BlOcks of Missing values): a new technique that reconstructs shapes, amplitudes and width of missing peaks and valleys in irregular time series. REBOM successfully reconstructs peaks and valleys by iteratively considering the time series itself and its correlation to multiple other time series. We provide an iterative algorithm to recover blocks of missing values and analytically investigate its monotonicity and termination. Our experiments with synthetic and real world hydrological data confirm that for the recovery of blocks of missing values in irregular time series REBOM is more accurate than existing methods.

Keywords

Missing blocks recovery, irregular time series, Singular Value Decomposition, ranking matrix.

1. INTRODUCTION

Time series data arise in a variety of domains, such as environmental, telecommunication, financial, and medical data. For example, in the field of hydrology, sensors are used to capture environmental phenomena including temperature, air pressure, and humidity at different points in time. For such data, it is not uncommon that more than 20% of the data is missing as blocks, i.e., multiple consecutive measurements are missing.

Existing techniques effectively recover blocks of missing values in regular time series, i.e., time series series containing peaks and valleys with a possibly varying frequency or amplitude that follow one or more periodic models, e.g., the sinus model where the frequency varies over time. The recovery accuracy of these techniques decreases for irregular time series, i.e., time series containing peaks and valleys that do not follow any model. In this work, we address the problem of finding the optimal recovery of blocks of missing values in irregular time series. We propose REBOM (REcovery of BlOcks of Missing values), a new data driven recovery technique for blocks of missing values that is able to restore missing peaks and valleys. We use the correlation [1] between time series to recover blocks of missing values. Intuitively, time series that tend to change their peaks and valleys simultaneously are correlated and we use the Pearson coefficient to quantify this correlation.

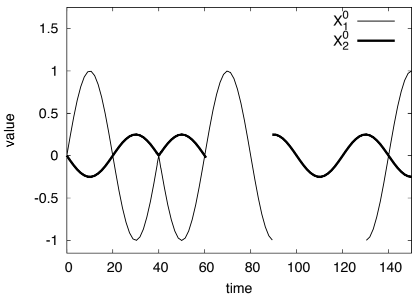
REBOM is an iterated low rank Singular Value Decomposition (SVD) [2]. We decompose a matrix $\mathbf{V}$ of correlated time series, where missing values have been initialized through linear interpolation combined with nearest neighbor imputation, into the product $\mathbf{L} \times \mathbf{\Sigma} \times \mathbf{R}^T$ of three matrices. By nullifying the smallest singular value of $\mathbf{\Sigma}$ we give higher priority to the correlation between the time series. The subsequent matrix multiplication yields an approximation of $\mathbf{V}$ that better approximates the missing values. After each iteration, the ranking of the most correlated time series with respect to the time series to recover, is updated. The iterative recovery terminates if the total ranking, which is determined by considering all observations of the time series, is identical to the partial ranking, which is determined by considering only observations with timestamps of missing values. If the total and the partial ranking are equal, the correlation can no longer be used to improve the recovery of missing values.

Problem definition: Assume a set of n irregular correlated time series $\mathbf{X}^0 = \{X_1^0, X_2^0, \ldots, X_n^0\}$ where $X_1^0, X_2^0, \ldots, X_n^0$ contain blocks of missing values. We propose a recovery method that determines, in j iterations, a set of time series $\widetilde{\mathbf{X}}^j = \{\widetilde{X}_1^j, \widetilde{X}_2^j, \ldots, \widetilde{X}_n^j\}$ where the missing blocks of $X_1^0, X_2^0, \ldots, X_n^0$ have been restored.

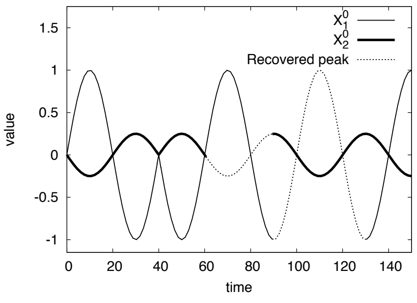
The result of REBOM for the recovery of peaks and valleys for two correlated time series is illustrated in Figure 1. Each time series is displayed as a 2d plot where the x-axis shows the timestamp t and the y-axis the value v for a given t. X_1^0 represents an air pressure time series and contains a missing block for the time range $]90, 130[$. X_2^0 represents a temperature time series that contains a missing block for the time range $]60, 90[$. REBOM can be used to restore the missing blocks of X_1^0 and X_2^0.

Figure 1 illustrates that REBOM accurately recovers *shape*, *amplitude* and *width* of the missing blocks. REBOM

The 18th International Conference on Management of Data (COMAD), 14th-16th Dec, 2012 at Pune, India.



(a) Original Time Series



(b) Restoration of Missing Blocks of X_1^0 and X_2^0

Figure 1: Recovery Performed by REBOM

detects that the peaks and valleys of X_1^0 and X_2^0 are correlated (high pressure corresponds to low temperature and vice versa). The shape and the width of the missing block are recovered from the position of the local extrema of X_1^0 with respect to the local extrema of the correlated time series X_2^0. The amplitude of the missing block of X_1^0 is recovered based on the two preceding peaks of X_1^0.

At the technical level, we show how to iterate the low rank SVD and we analytically investigate the main properties of the method. The main contributions of this paper are:

- We propose REBOM: an iterated low rank SVD that iteratively refines the initial recovery of missing values.
- We propose a greedy algorithm that repeatedly selects a time series with missing values that have been initialized and uses the k most correlated time series to iteratively refine the recovery of the missing values.
- We prove that our greedy algorithm is stepwise monotonic, i.e., the accuracy of the recovery increases by choosing, at each step, the most correlated time series. The algorithm terminates when the set of the most correlated time series does not change anymore.
- We empirically show that the recovery accuracy of REBOM is invariant to the initial recovery. Different initialization methods lead to the same recovery accuracy but with different number of iterations.
- We present an experimental evaluation of the accuracy of our technique that compares REBOM to state-of-the-art techniques for the recovery of blocks of missing values. The results show the superiority of our algorithm for the restoration of peaks and valleys.

The rest of the paper is organized as follows. Section 2 reviews related work on reduction methods and existing techniques for imputing missing values. Section 3 defines the initialization method and describes the basics of the low rank SVD. Section 4 introduces and discusses REBOM and its properties. Section 5 empirically compares the results of REBOM to other techniques proposed in the literature for the recovery of blocks of missing values.

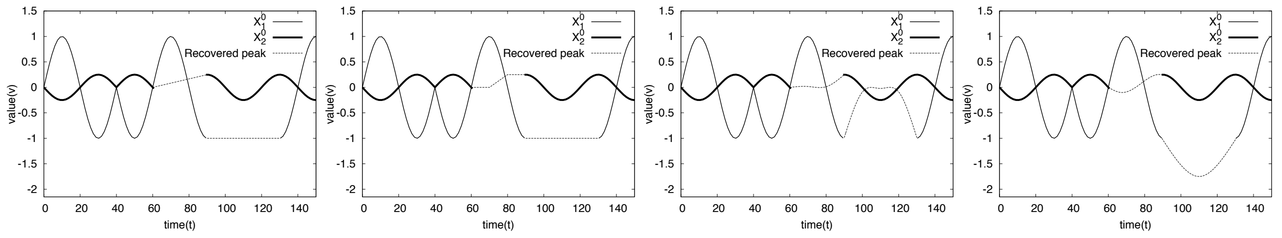
2. RELATED WORK

Prediction models such as Maximum Likelihood Estimation (MLE) [3], Bayesian Networks (BN) [4, 5] and Expectation Maximization (EM) [6] were used to estimate single missing values or small blocks of missing values in time series. These techniques are parametric and require a specific type of data distribution, e.g, Gaussian distribution. Therefore, they only perform well for the recovery of blocks of missing values in regular time series where peaks and valleys follow a periodic model of constant frequency and amplitude.

Li et al. [7] presented an approach called DynaMMo that is based on Expectation Maximization (EM) and Kalman Filter [8]. This technique is intended to recover missing blocks in non linear time series that contain peaks and valleys. DynaMMo allows to use one reference time series in addition to the time series that contains the missing block. The Kalman Filter uses the data of the time series that contains missing blocks together with a reference time series, to estimate the current state of the missing blocks. This estimation is performed as a multi step process that uses two different estimators. The first estimator represents the current state and the second estimator represents the initial state and the error of the estimation. For every step of the process, an EM method predicts the value of the current state and then the two estimators are used to refine the predicted values of the current state and to maximize their likelihood. DynaMMo does not allow to use more than one reference time series for the block recovery. DynaMMO performs an accurate block recovery for any type of regular time series. The accuracy of the block recovery decreases for irregular time series (cf. Section 5).

Techniques that rely on basic statistical methods such as mean imputation, piecewise approximation (linear spline, cubic spline, ...) [9, 10], regression [11, 12] and k Nearest Neighbors [13, 14] have been proposed for the recovery of blocks of missing values. Figures 2(a) and 2(b) illustrate the block recovery performed respectively by linear spline and k nearest neighbor using values at t=60 and t=90. Figure 2(c) shows that the regression method replaces missing values by points lying on the line that minimizes the regression error of all existing points. These techniques are not able to accurately recover any of the two missing blocks in X_1^0 and X_2^0. The cubic spline technique finds a third order polynomial that connects three successive values. Figure 2(d) shows that the cubic spline replaces the missing block by a block opposite to the one that precedes the missing block. Cubic spline is able to perform a good recovery only for the missing block of X_1^0. All basic methods are not suitable techniques for block recovery in regular time series where peaks and valleys follow a periodic model of varying amplitude or frequency, or in irregular time series.

Kurucz et al. [15] proposed a technique based on EM and Singular Value Decomposition (SVD) [16, 17, 18, 19] for



(a) Linear Spline Recovery (b) k Nearest Neighbor Recovery (c) Regression Recovery (d) Cubic Spline Recovery

Figure 2: Recovery using Different Techniques

comparing recommender systems where one of them contains missing values. A recovery of the missing values is performed before the comparison process. Each recommender system is represented by one column of values in a rating matrix which is decomposed using SVD. The result of the decomposition is modified using a method called gradient boosting [20]. The EM algorithm is then applied to refine the result of gradient boosting. The proposed solution dynamically discovers data dependencies from coordinate axes that represent the recommender systems and is applicable for more than one reference recommender system. However, the application of gradient boosting on different recommender systems looses the dependencies among the original values of recommender systems. Therefore, this technique yields bad results for block recovery in case where more than one recommender system contains missing blocks.

Tree-based methods were proposed to impute missing values. He [21] and Ding and Simonoff [22] present an overview of tree classification methods that are able to replace missing values in time series. These trees find the optimal way to classify missing values using a regression approach and are called Classification and Regression Trees (CART). These techniques are designed to create a classification of the missing values. Missing values that belong to the same class will be recovered with the same value. Therefore, these methods are not able to effectively restore missing peaks and valleys in regular and irregular time series.

3. PRELIMINARIES AND BACKGROUND

3.1 Notation

We use the following notation: sets and vectors are upper-case, matrices are upper-case bold, and elements of sets and matrices are lower-case. A time series $X_1 = \{x_1, x_2, \ldots, x_n\}$ is a set of n observations. Each observation x_j from X_1 is a pair (t_j, v_j) where t_j and v_j are respectively the timestamp and the value of the observation. $T_1 = \{t \mid (t, \_) \in X_1)\}$ denotes the set of all timestamps from X_1; $V_1 = \{v \mid (\_, v) \in X_1)\}$ denotes the vector of all values from the time series X_1. A time series X_1 with missing values that have not been recovered yet, is denoted as X_1^0.

3.2 Preprocessing of Time Series

The first preprocessing step uses basic statistical methods to initialize all missing values. After the initialization the timestamps of all time series are aligned.

DEFINITION 1 (MISSING TIMESTAMPS). *Given a set of n time series $\{X_1^0, \ldots, X_n^0\}$, the set of missing timestamps of time series X_i^0 with respect to the timestamps of the other time series is $T_i^0 = \{t \mid ((t, \_) \in X_1^0 \vee \ldots \vee (t, \_) \in X_n^0) \wedge (t, \_) \notin X_i^0\}$.*

Note that missing timestamps of one time series have to be present in at least another time series. Timestamps missing in all time series are not considered. An additional preprocessing step can be added if such timestamps shall be recovered as well.

$X_1^1 = \{(t_1, v_1), \ldots, (t_n, v_n)\}$ is the initial recovery of X_1^0 iff $\forall i \in \{1, \ldots, n\}$

$$(t_i, v_i) = \begin{cases} (t_i, v_i) \textbf{ if } (t_i, v_i) \in X_1^0 \\ \textbf{Else} \begin{cases} (t_i, v) \textbf{ if } (s(t_i), \_) \notin X_1^0, \\ \qquad (p(t_i), v) \in X_1^0 \\ (t_i, v) \textbf{ if } (p(t_i), \_) \notin X_1^0, \\ \qquad (s(t_i), v) \in X_1^0 \\ (t_i, \frac{(t_i - p(t_i))(s(v_i) - p(v_i))}{s(t_i) - p(t_i)} + s(v_i)) \\ \qquad \textbf{otherwise} \end{cases} \end{cases}$$

$p(t_i) = max\{t_j \mid (t_j, \_) \in X_1^0 \wedge t_j < t_i\}$ is the predecessor of timestamp t_i in X_1^0 and $s(t_i) = min\{t_j \mid (t_j, \_) \in X_1^0 \wedge t_j > t_i\}$ is the successor timestamp of t_i in X_1^0. Similarly, $p(v_i) = \{v_j \mid (t_j, \_) \in X_1^0 \wedge t_j = p(t_i)\}$ is the predecessor of value v_i in X_1^0 and $s(v_i) = \{t_j \mid (t_j, \_) \in X_1^0 \wedge t_j = s(t_i)\}$ is the successor value of v_i in X_1^0. Thus, the initial recovery of the missing values is a linear interpolation. If the missing values occur as the first or the last elements of X_1^0, we use the nearest neighbor imputation.

Two time series X_1^1 and X_2^1 with initialized missing values define a set of multidimensional points: $\{(v, v') \mid (t, v) \in X_1 \wedge (t, v') \in X_2\}$. The second preprocessing step constructs a matrix with n m-dimensional points from m time series with n observation each.

EXAMPLE 1. *Figure 3 shows two time series X_1^0 and X_2^0 with missing values, the initialized time series X_1^1 and X_2^1, and the set of multidimensional points $\mathbf{V}$. The initialized missing values are highlighted in gray.*

From Definition 1 we get $T_1^0 = \{100, 110, 120\}$ and $T_2^0 = \{70, 80\}$.

3.3 Low Rank Matrix Decomposition

3.3.1 Singular Value Decomposition

The Singular Value Decomposition (SVD) is a matrix decomposition method that decomposes a matrix $\mathbf{V}$ into three matrices $\mathbf{L}$, $\mathbf{\Sigma}$ and $\mathbf{R}^T$. The product of the three matrices is equal to $\mathbf{V}$.

$\mathbf{X}_1^0$	
t	v
0	0
10	1
20	0
30	-1
40	0
50	-1
60	0
70	1
80	-1
90	-1
130	-1
140	0
150	1

$\mathbf{X}_2^0$	
t	v
0	0
10	-0.25
20	0
30	0.25
40	0
50	0.25
60	0
90	0.25
100	0
110	-0.25
120	0
130	0.25
140	0
150	-0.25

$\Rightarrow$

$\mathbf{X}_1^1$	
t	v
0	0
10	1
20	0
30	-1
40	0
50	-1
60	0
70	1
80	-1
90	-1
100	-1
110	-1
120	-1
130	-1
140	0
150	1

$\mathbf{X}_2^1$	
t	v
0	0
10	-0.25
20	0
30	0.25
40	0
50	0.25
60	0
70	0.08
80	0.16
90	0.25
100	0
110	-0.25
120	0
130	0.25
140	0
150	-0.25

$\Rightarrow$

$\mathbf{V}$	
V_1	V_2
0	0
1	-0.25
0	0
-1	0.25
0	0
-1	0.25
0	0
1	0.08
-1	0.16
-1	0.25
-1	0
-1	-0.25
-1	0
-1	0.25
0	0
1	-0.25

Figure 3: Original Time Series X_1^0, X_2^0; Initialized Time Series X_1^1, X_2^1; Multidimensional Points V

DEFINITION 2 (SVD). *A matrix $\mathbf{V} = [V_1|V_2|\ldots|V_n] \in \mathcal{R}^{m\times n}$ can be decomposed into a product of three matrices:*

$$\begin{aligned} SVD(\mathbf{V}) &= \mathbf{L} \times \mathbf{\Sigma} \times \mathbf{R}^T \\ &= \underbrace{\left[L_1 \middle| \ldots \middle| L_n \right]}_{\mathbf{L}(m\times n)} \times \underbrace{\left[\begin{array}{c|c|c} \sigma_1 & \ldots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \ldots & \sigma_n \end{array}\right]}_{\mathbf{\Sigma}(n\times n)} \times \underbrace{\left[\begin{array}{c} R_1^T \\ \hline \vdots \\ \hline R_n^T \end{array}\right]}_{\mathbf{R}^T(n\times n)} \end{aligned}$$

Where:

1. *$\mathbf{\Sigma}$: is a $n \times n$ square diagonal matrix that contains strictly positive singular values of $\mathbf{V}$. The diagonal entries σ_i of $\mathbf{\Sigma}$ are the square roots of the eigen values of $\mathbf{V}^T\mathbf{V}$ and are ranked in decreasing order such that $\sigma_1 > \sigma_2 > \ldots > \sigma_n$.*
2. *$\mathbf{L}$: is an $m \times n$ orthogonal matrix whose columns are the orthonormal eigen vectors of $\mathbf{V}\mathbf{V}^T$ ($L^T L = I$, where I is the identity matrix). The eigen vectors of L are computed by solving $Det(\sigma\mathbf{I} - \mathbf{V}\mathbf{V}^T) = 0$ where $Det(\mathbf{X})$ is the determinant of matrix $\mathbf{X}$.*
3. *$\mathbf{R}$: is an $n \times n$ orthogonal matrix having as columns orthonormal eigen vectors of $\mathbf{V}^T\mathbf{V}$ ($R^T R = I$). The eigen vectors of $\mathbf{R}$ are computed by solving $Det(\sigma\mathbf{I} - \mathbf{V}^T\mathbf{V}) = 0$.*
4. *A singular value σ_i defines the variance of vector L_i along dimension R_i^T. Each dimension represents an axis of projection: $var(L_i) = \sigma_i$.*

EXAMPLE 2. *Consider time series X_1^1 and X_2^1 from Figure 3. Figure 4 illustrates the SVD of $\mathbf{V}$.*

3.3.2 Dimensionality Reduction

SVD allows to perform a dimensionality reduction from a dimension n to a lower dimension r. The dimensionality reduction is performed by nullifying the $n-r$ smallest singular values from matrix $\mathbf{\Sigma}$, where $0 < \sigma_r < \sigma_n$. Figure 5 illustrates the dimensionality reduction for $r = n-1$, i.e., the smallest singular value of $\mathbf{\Sigma}$ is nullified. We write $SVD_r(\mathbf{V})$ for the result of a low rank SVD of a matrix $\mathbf{V}$. REBOM uses the low rank SVD for improving the initial imputation of the missing values as described in the next section.

$$SVD(\mathbf{V}) =$$

$$\underbrace{\begin{bmatrix} 0 & 0 \\ 0.31 & -0.22 \\ 0 & 0 \\ -0.31 & 0.22 \\ 0 & 0 \\ -0.31 & 0.22 \\ 0 & 0 \\ -0.30 & -0.11 \\ -0.30 & 0.04 \\ -0.31 & 0.22 \\ -0.30 & -0.27 \\ -0.30 & -0.75 \\ -0.30 & -0.27 \\ -0.31 & 0.22 \\ 0.00 & 0.00 \\ 0.31 & -0.22 \end{bmatrix}}_{\mathbf{L}} \times \underbrace{\begin{bmatrix} 3.35 & 0 \\ 0 & 0.51 \end{bmatrix}}_{\mathbf{\Sigma}} \times \underbrace{\begin{bmatrix} 0.99 & -0.14 \\ 0.14 & 0.99 \end{bmatrix}}_{\mathbf{R}^T}$$

Figure 4: Example of Singular Value Decomposition

$$SVD_r(\mathbf{V}) =$$

$$\underbrace{\left[L_1 \middle| \ldots \middle| L_m \right]}_{\mathbf{L}(m\times n)} \times \underbrace{\left[\begin{array}{c|c|c|c} \sigma_1 & \ldots & 0 & 0 \\ \hline \vdots & \ddots & \vdots & 0 \\ \hline 0 & \ldots & \sigma_r & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]}_{\mathbf{\Sigma_r}(n\times n)} \times \underbrace{\left[\begin{array}{c} R_1^T \\ \hline \vdots \\ \hline R_n^T \end{array}\right]}_{\mathbf{R}^T(n\times n)}$$

Figure 5: Illustration of Dimensionality Reduction

4. REBOM

REBOM combines the characteristics of a time series with missing values with the characteristics of its most correlated time series to recover blocks of missing values in irregular time series.

4.1 Correlation Ranking Matrix

We define the top-k ranking matrix to capture the correlation between different time series. The correlation is defined over all values of the first vector of the matrix with respect to all values of another vector. The Pearson coefficient is used as a correlation metric. Given two vectors $V_i = [v_{i_1}, v_{i_2}, \ldots, v_{i_n}]$ and $V_j = [v_{j_1}, v_{j_2}, \ldots, v_{j_n}]$ of the same length n, the Pearson correlation coefficient ρ of V_i with respect to V_j is defined as follows:

$$\begin{aligned} \rho(V_i, V_j) &= \frac{cov(V_i, V_j)}{\sqrt{var(V_i)var(V_j)}} \\ &= \frac{\sum_{p=1}^{n}(v_{i_p} - \bar{v}_i)(v_{j_p} - \bar{v}_j)}{\sqrt{\sum_{p=1}^{n}(v_{i_p} - \bar{v}_i)^2 \sum_{p=1}^{n}(v_{j_p} - \bar{v}_j)^2}} \\ & \quad \textit{with } \bar{v}_i = \frac{1}{n}\sum_{p=1}^{n} v_{i_p}, \quad \bar{v}_j = \frac{1}{n}\sum_{p=1}^{n} v_{j_p} \end{aligned}$$

$\rho(V_i, V_j)$ is undefined if all values of V_i or V_j are equal. The vectors of the correlation ranking matrix are ranked in decreasing order of the Pearson coefficient between the first vector and the remaining vectors.

DEFINITION 3 (TOP-K RANKING MATRIX). *Let $\mathbf{V} = [V_1, V_2, \ldots, V_n]$ be a matrix of n vectors. $\mathbf{V}^{top\text{-}k} =$*

$[V_1', V_2', \dots, V_k']$ is defined as the top-k ranking matrix of $\mathbf{V}$ *with respect to a given vector that contains initialized missing values $V_q^1 \in \mathbf{V}$ iff:*

- *$\mathbf{V}^{top\text{-}k}$ contains the k vectors that are most correlated to V_q^1: $\forall V_i' \in \mathbf{V}^{top\text{-}k} \forall V_j \in \mathbf{V} \setminus \mathbf{V}^{top\text{-}k} : |\rho(V_i', V_q^1)| \geq |\rho(V_j, V_q^1)|$*
- *The elements of $\mathbf{V}^{top\text{-}k}$ are sorted by their correlation coefficient to V_q^1 : $\forall 1 \leq i < k : |\rho(V_i', V_q^1)| \geq |\rho(V_{i+1}', V_q^1)|$*

For each matrix $\mathbf{V}^{top\text{-}k}$ we define a corresponding top-k ranking vector $\rho_{\mathbf{V}^{top\text{-}k}} = [\rho(V_q^1, V_1^{top\text{-}k}), \rho(V_q^1, V_2^{top\text{-}k}), \dots, \rho(V_q^1, V_k^{top\text{-}k})]$ for V_q^1 with the l_1-norm $||\rho_{\mathbf{V}^{top\text{-}k}}|| = \sum_{i=1}^{k}(|\rho(V_q^1, V_i^{top\text{-}k})|)$.

Example 3. *Consider Figure 6 with $\mathbf{V} = [V_1, V_2, V_3, V_4]$ and top-3 ranking $\mathbf{V}^{top\text{-}3} = [V_4, V_3, V_1]$ for V_4.*

$$\mathbf{V} = \begin{bmatrix} 4 & 6 & 3 & 2 \\ 5 & 7 & 1 & 3 \\ 6 & 7 & 9 & 8 \\ 7 & 6 & 8 & 7 \end{bmatrix}, \mathbf{V}^{top\text{-}3} = \begin{bmatrix} 2 & 3 & 4 \\ 3 & 1 & 5 \\ 8 & 9 & 6 \\ 7 & 8 & 7 \end{bmatrix}$$

Figure 6: Example of $\mathbf{V}^{top\text{-}3}$

We get $\rho_{\mathbf{V}^{top\text{-}3}} = [\rho(V_4, V_4), \rho(V_4, V_3), \rho(V_4, V_1)] = [1, 0.93, 0.87]$ and $||\rho_{\mathbf{V}^{top\text{-}3}}|| = 2.75$.

4.2 Stepwise Correlation Monotonicity

We prove that REBOM is stepwise monotonic, i.e, choosing a bigger correlation value in the same iteration implies a bigger sum of variances. Lemma 1 states that the l_1-norm of a ranking vector $\rho_{\mathbf{V}}$ is proportional to the sum of the variance of vectors obtained by the application of the low rank SVD. In what follows a submatrix $\mathbf{V}_i = [V_{i_1}, V_{i_2}, \dots, V_{i_k}]$ that contains k different columns of $\mathbf{V}$ is denoted as $\mathbf{V}_i \in \mathbf{V}$.

Lemma 1. *Let $\mathbf{V}_i = [V_{i_1}, V_{i_2}, \dots, V_{i_k}]$ and $\mathbf{V}_j = [V_{j_1}, V_{j_2}, \dots, V_{j_k}]$ be two different $m \times k$ matrices and let $\mathbf{V}$ be $m \times n$ matrix such that $n \geq k$ and $\mathbf{V}_i, \mathbf{V}_j \in \mathbf{V}$. Let $\mathbf{W}_i = [W_{i_1}, W_{i_2}, \dots, W_{i_k}] = SVD_r(\mathbf{V}_i)$ and $\mathbf{W}_j = [W_{j_1}, W_{j_2}, \dots, W_{j_k}] = SVD_r(\mathbf{V}_j)$ such that $V_{i_1} = V_{j_1}$. The l_1-norm of $\rho_{\mathbf{V}_i}$ and $\rho_{\mathbf{V}_j}$ is proportional to the sum of the variances of $\mathbf{W}_i$ and $\mathbf{W}_j$:*

$$||\rho_{\mathbf{V}_i}|| > ||\rho_{\mathbf{V}_j}|| \Rightarrow \sum_{p=1}^{k} var(W_{j_p}) > \sum_{p=1}^{k} var(W_{i_p})$$

Lemma 1 states that choosing a matrix with a bigger l_1-norm of the ranking vector implies a higher sum of variances over the vectors obtained by the SVD. Therefore, more correlated vectors of the input matrix yields a higher sum of the variances after the application of $SVD_r()$. Thus, by considering the top-k ranking matrix, the result of $SVD_r()$ maximizes the following objective function:

$$\sum_{V_i \in SVD_r(\mathbf{V})} var(V_i)$$

$$\mathbf{V} = \begin{bmatrix} 4 & 6 & 3 & 2 \\ 5 & 7 & 1 & 3 \\ 6 & 7 & 9 & 8 \\ 7 & 6 & 8 & 7 \end{bmatrix}, \mathbf{W} = \begin{bmatrix} 4.2 & 5.6 & 2.3 & 2.8 \\ 4.9 & 7.1 & 1.4 & 2.4 \\ 6.6 & 6.6 & 8.9 & 7.7 \\ 6.2 & 6.4 & 8.1 & 7.1 \end{bmatrix}$$

Figure 7: Example of a matrix and its SVD_r transformation

Example 4. *Consider matrix $\mathbf{V} = V_1, V_2, V_3, V_4$ from example 3 and the result matrix of the application of $SVD_r(\mathbf{V})$ as shown in Figure 7.*

Let's take the example of $\mathbf{V}_1, \mathbf{V}_2 \in \mathbf{V}$ where $\mathbf{V}_1 = \{V_4, V_3, V_1\}$ and $\mathbf{V}_2 = \{V_4, V_2, V_1\}$, and let $\mathbf{W}_1 = SVD_r(\mathbf{V}_1)$ and $\mathbf{W}_2 = SVD_r(\mathbf{V}_2)$.

If we apply the computation with respect to vector V_4, we get $||\mathbf{V}_1|| = 2.75, ||\mathbf{V}_2|| = 2.07, \sum_{p=3}^{k} var(W_{1_p}) = 24$ and $\sum_{p=1}^{3} var(W_{2_p}) = 9.4$.

Lemma 1 holds for any other matrices $\mathbf{V}_i, \mathbf{V}_j \in \mathbf{V}$.

4.3 Iterative Recovery of REBOM

This section proves that REBOM terminates. In each step we compute the partial correlation ranking for the time series based on the missing values. If this partial ranking is the same as the global ranking, the recovery stops. For all missing values $t \in T_i^0$ (cf. Definition 1) the partial correlation $\widetilde{\rho}(V_i, V_j)$ is defined as follows:

$$\widetilde{\rho}(V_i, V_j) = \frac{\sum_{t=1}^{|T_i^0|}(v_{i_t} - \bar{v}_i)(v_{j_t} - \bar{v}_j)}{\sqrt{\sum_{t=1}^{|T_i^0|}(v_{i_t} - \bar{v}_i)^2 \sum_{t=1}^{|T_i^0|}(v_{j_t} - \bar{v}_j)^2}}$$

Where $|T_i^0|$ is the length of T_i^0. $\widetilde{\rho}(V_i, V_j)$ is undefined if all missing values of V_i or V_j are equal. The partial ranking matrix contains the partially most correlated vectors to the vector that contains the missing blocks to recover.

Definition 4 (Partial ranking matrix). *Given a matrix $\mathbf{V} = [V_1, V_2, \dots, V_n]$ of n vectors, $\widetilde{\mathbf{V}}^{top\text{-}k} = [V_1', V_2', \dots, V_k']$ is defined as the top-k partial ranking matrix of $\mathbf{V}$ with respect to a given vector $V_q^1 \in \mathbf{V}$ iff:*

- *$\widetilde{\mathbf{V}}^{top\text{-}k}$ contains the k vectors that are partially most correlated to V_q^1: $\forall V_i' \in \widetilde{\mathbf{V}}^{top\text{-}k} \forall V_j \in \mathbf{V} \setminus \widetilde{\mathbf{V}}^{top\text{-}k}$: $\widetilde{\rho}(V_i', V_q^1) \geq \widetilde{\rho}(V_j, V_q^1)$*
- *The elements of $\widetilde{\mathbf{V}}^{top\text{-}k}$ are sorted by their partial correlation coefficient to V_q^1 : $\forall 1 \leq i < k : \widetilde{\rho}(V_i', V_q^1) \geq \widetilde{\rho}(V_{i+1}', V_q^1)$*

The top-k ranking and the top-k partial ranking are used to terminate the iterative recovery process.

Lemma 2 (Termination Condition). *Let $\mathbf{W}_i^{top\text{-}k} = [W_{i_1}, W_{i_2}, \dots, W_{i_k}]$ and let Ranking() be the ranking of vectors inside a matrix. If $\mathbf{W}_i^{top\text{-}k}$ and its partial correlation matrix have the same ranking then the algorithm can not anymore create a matrix $\mathbf{W}_{i+1}$ with bigger sum of variances along its vectors. Formally:*

$$Ranking(\mathbf{W}_i^{top\text{-}k}) = Ranking(\widetilde{\mathbf{W}}_i^{top\text{-}k}) \Rightarrow$$

$$\sum_{W_{i_j} \in \mathbf{W_i}} var(W_{i_j}) > \sum_{W_{(i+1)_j} \in \mathbf{W}_{i+1}} var(W_{(i+1)_j})$$

After each iteration, REBOM compares the ranking of vectors in the top-k ranking with the ranking of vectors in the top-k partial ranking. If the two rankings are equal, the recovery process terminates. As long as the two rankings are different or one of the two rankings is undefined, the most correlated time series can be used to further improve the accuracy of the recovery.

EXAMPLE 5. *Let $\mathbf{V}_1 = [V_{1_1}, V_{1_2}, V_{1_3}, V_{1_4}, V_{1_5}]$ and $k = 3$. After each iteration we create matrix $\mathbf{W}_i$ with recovered values and compare $Ranking(\mathbf{W}^{top\text{-}3})$ with $Ranking(\widetilde{\mathbf{W}}^{top\text{-}3})$. Initially, $\widetilde{\rho}(V_1, V_i)$ and $Ranking(\widetilde{\mathbf{V}}^{top\text{-}3})$ are undefined and thus, REBOM iterates. REBOM terminates after two steps since $Ranking(\mathbf{W}_2^{top\text{-}3}) = Ranking(\widetilde{\mathbf{W}}_2^{top\text{-}3}) = \{V_1, V_2, V_3\}$. The vectors of the top-k ranking and top-k partial ranking are highlighted in gray and the recovered values are displayed in bold.*

	$\mathbf{V}_1$				
	V_{1_1}	V_{1_2}	V_{1_3}	V_{1_4}	V_{1_5}
	-1	0.5	0.25	0.75	1
	0	0	0.2	0	0
	-1	0.5	0.25	0	1
	0	0.5	0	0.75	0
	1	0	0	0	0
	0	0	-0.25	0	0
	-1	0.5	0.25	0.75	-1
	-1	0.2	0	0	0.7
	-1	0.4	-0.25	0	0.4
	-1	0.2	0	0.75	0.8
	-1	0.5	0.25	0.75	1
	0	0	0	0	0
	-1	0.5	0.25	0.75	1
$\rho(V_{1_1}, V_{1_i})$	1	-0.69	-0.33	-0.43	-0.46
$\widetilde{\rho}(V_{1_1}, V_{1_i})$	-	-	-	-	-

	$\mathbf{W}_1$				
	W_{1_1}	W_{1_2}	W_{1_3}	W_{1_4}	W_{1_5}
	-1	0.5	0.25	0.75	1
	0	0	0.2	0	0
	-1	0.5	0.25	0	1
	0	0.5	0	0.75	0
	1	0	0	0	0
	0	0	-0.25	0	0
	-1	0.5	0.25	0.75	-1
	-0.5	0.2	0	0	0.7
	-0.8	0.4	-0.25	0	0.4
	-0.5	0.2	0	0.75	0.8
	-1	0.5	0.25	0.75	1
	0	0	0	0	0
	-1	0.5	0.25	0.75	1
$\rho(W_{1_1}, W_{1_i})$	1	-0.78	-0.45	-0.47	-0.41
$\widetilde{\rho}(W_{1_1}, W_{1_i})$	1	-1	1	0.5	0.97

	$\mathbf{W}_2$				
	W_{2_1}	W_{2_2}	W_{2_3}	W_{2_4}	W_{2_5}
	-1	0.5	0.25	0.75	1
	0	0	0.2	0	0
	-1	0.5	0.25	0	1
	0	0.5	0	0.75	0
	1	0	0	0	0
	0	0	-0.25	0	0
	-1	0.5	0.25	0.75	-1
	-0.2	0.2	0	0	0.7
	-0.8	0.4	-0.25	0	0.4
	-0.2	0.2	0	0.75	0.8
	-1	0.5	0.25	0.75	1
	0	0	0	0	0
	-1	0.5	0.25	0.75	1
$\rho(W_{2_1}, W_{2_i})$	1	-0.8	-0.48	-0.46	-0.36
$\widetilde{\rho}(W_{2_1}, W_{2_i})$	1	-1	1	0.5	0.97

Figure 8: Iterative Recovery of REBOM

4.4 Algorithm

Algorithm 1 implements the block recovery of REBOM. First, using the method described in subsection 3.2, $\mathbf{X}^1$ is created by initializing the missing values of $\mathbf{X}^0$. Then, the vectors representing each time series of $\mathbf{X}^1$ are inserted as columns in the matrix of vectors $\mathbf{W}_1$. The vector to recover is inserted as the first column of $\mathbf{W}_1$. The order of the selected vector to recover has no impact on the result of the recovery since only the original vectors are used in the recovery process. Therefore, the proposed recovery is deterministic and does not depend on the order of time series to recover. Next, if the ranking of the top-k ranking matrix is different from the ranking of the top-k partial matrix or one of the rankings is undefined (NAN), the recovery is performed. If $Ranking(\mathbf{W}_j^{top\text{-}k})$ is equal to $Ranking(\widetilde{\mathbf{W}}_j^{top\text{-}k})$ the recovered time series is inserted into the set of recovered time series, i.e, $\widetilde{\mathbf{X}}^j$. Once all time series have been recovered, $\widetilde{\mathbf{X}}^j$ will be returned as the result of REBOM's block recovery.

```
Input: A set of n time series
       $\mathbf{X}^0 = \{X_1^0, X_2^0, \ldots, X_n^0\}$
Output: A set of recovered time series
        $\widetilde{\mathbf{X}} = \{\widetilde{X}_1^{j_1}, \widetilde{X}_2^{j_2}, \ldots, \widetilde{X}_n^{j_n}\}$
begin
    $\mathbf{X}^1 = Init(\mathbf{X}^0)$;
    for each $X_i^1 \in \mathbf{X}^1$ do
        $V_i^1 = Extract\_val(X_i^1)$;
        $j = 1$;
        $\mathbf{W}_j = [V_i^1]$;
        for each $\widetilde{X}_p^1 \in \widetilde{\mathbf{X}}^1 \setminus \widetilde{X}_i^1$ do
            $\mathbf{V}_p^1 = Extract\_val(\widetilde{X}_p^1)$;
            $\mathbf{W}_j = [\mathbf{W}_j, V_p^1]$;
        while
        $Ranking(\mathbf{W}_j^{top\text{-}k}) <> Ranking(\widetilde{\mathbf{W}}_j^{top\text{-}k})$ or
        $Ranking(\mathbf{W}_j^{top\text{-}k}) = NAN$ or
        $Ranking(\widetilde{\mathbf{W}}_j^{top\text{-}k}) = NAN$ do
            $\mathbf{L\Sigma R}^T = SVD(\mathbf{W}_j^{top\text{-}k})$;
            $\mathbf{\Sigma}_r = Reduce\_Dim(\mathbf{\Sigma}, n, r)$;
            $\mathbf{M} = \mathbf{L} \times \mathbf{\Sigma}_r \times \mathbf{R}^T$;
            $\mathbf{W}_j = UMV(\mathbf{W}_j^{top\text{-}k}, \mathbf{M})$;
            $j += 1$;
        $\widetilde{X}_i^j = Add\_ts(W_{j_i})$;
        $\widetilde{\mathbf{X}}^j = \{\widetilde{\mathbf{X}}^j\} \cup \{\widetilde{X}_i^j\}$;
        $i += 1$;
    return $\widetilde{\mathbf{X}}^j$;
end
```

Algorithm 1: REBOM's Block Recovery

Extract_val() and *Add_ts*() are used respectively to extract values from a time series and to add time stamps to a vector.

The UMV algorithm (cf. Algorithm 2) updates missing values. It accesses the database and uses procedural SQL to determine the indexes of missing values (load_mv_indexes()). The code of this function is described in the the first section of the appendix.

```
1 Algorithm:UMV($\mathbf{V}_1, \mathbf{V}_2$)
2 begin
3     for each $V_i \in \mathbf{V}_1$ do
4         $T_i^0$=load_mv_indexes($i$);
5         for each $v_{i_j} \in V_i$ do
6             if $position(v_{i_j}) \in T_i^0$ then
7                 Insert_element($\mathbf{V}_3, v'_{ij}$);
                  // Insert $v'_{ij} \in \mathbf{V}_2$ in row $i$ and
                      column $j$ of $\mathbf{V}_3$
8             else
9                 Insert_element($\mathbf{V}_3, v_{i_j}$);
10    return $\mathbf{V}_3$;
11 end
```
Algorithm 2: Updating Initialized Missing Values

5. EXPERIMENTS

5.1 Experimental Setup

For the evaluation we use real world datasets and synthetic data sets that describe hydrological phenomena of up to 15 million observations produced by sensors in 242 mountain stations. Our hydrological database contains 79 temperature time series, 69 precipitation time series, 48 water level time series, 15 humidity time series, 4 wind speed time series and 3 air pressure time series. The data was provided by an environmental engineering company [23].

We ran experiments to compare the recovery accuracy of REBOM against state-of-the-art techniques.

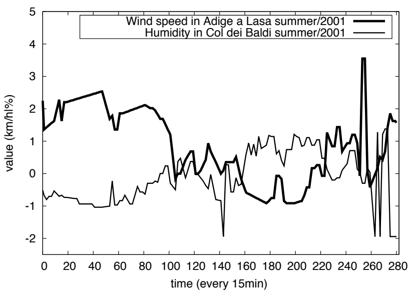
5.2 Experiments with Hydrological Time Series

5.2.1 Restoration of Peaks and Valleys

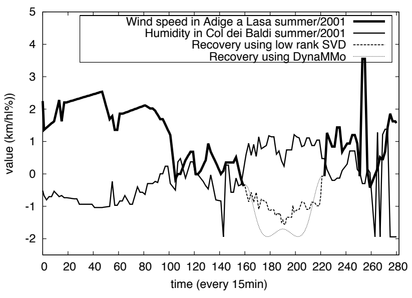
In the first set of experiments, we compare the accuracy of REBOM for the restoration of missing blocks against a non parametric recovery technique that is the (non-iterated) low rank SVD and a parametric recovery technique that is DynaMMo [10]. These two techniques are the most accurate techniques for the recovery of blocks of missing values in time series. We ran our experiment on wind speed and humidity time series. Figure 9(a) shows two time series measured during summer season (one measurement every 15 minutes) in two different areas of the region of Alto-Adige (Italy). We drop a block of values for $t \in]160, 220[$ and restore it using the low rank SVD and DynaMMo. The dropped block includes a valley with a small peak.

The recovery of the two techniques is shown in Figure 9(b). The low rank SVD is only able to detect part of the trend of the missing block, i.e., only a valley is recovered. The shape of the recovered valley resembles the shape of the block that belongs to the same time interval of the missing block in the other time series. DynaMMo is able to detect the entire trend of the missing block, i.e., a valley containing a small peak. However the shape of the original block is not accurately restored. The recovered block looks similar to a smooth spline that contains a small peak. Since we use only tow time series REBOM will not iterate. Therefore, the recovery of REBOM is similar to the recovery of the low rank SVD.

We add a second humidity time series to the experiment to



(a) Time Series Measured in Two Different Areas



(b) Recovery of a Removed Block

Figure 9: Recovery using Low Rank SVD and DynaMMo

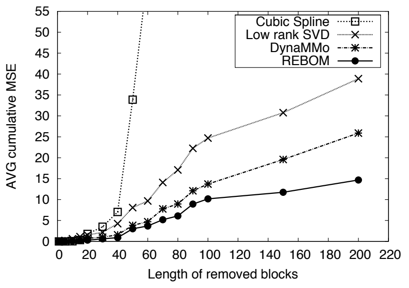
compare the block recovery of REBOM against DynaMMo (see Figure 10). The result of Figure 10(b) shows that the recovery of DynaMMo does not change by the addition of a third time series because DynaMMo cannot use more than one reference time series in the recovery process. REBOM exploits the two humidity time series in the recovery process. It uses the history of the wind speed time series together with the correlation with respect to the two humidity time series to recover the missing block. Both the trend and the shape of the missing block are accurately recovered. Adding more correlated time series will further improve the block recovery of REBOM (see Figure 12).

We run a second set of experiments in which we compare the block recovery error using the Mean Square Error (MSE):

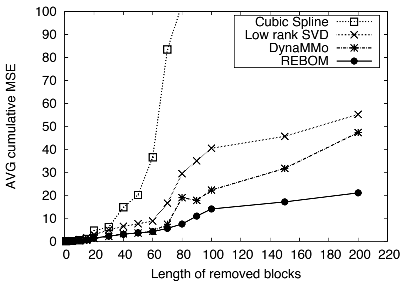
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(w_i - {v_i}^+)^2$$

where w is the recovered value, v^+ is the original value and n is the number of deleted observations.

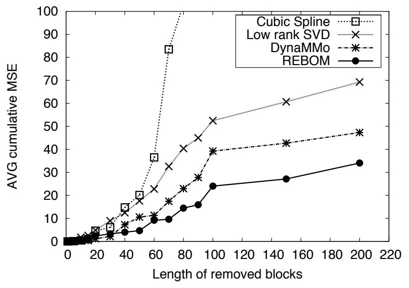
Figure 11 shows the cumulative recovery error for removed blocks of values of increasing length: we set a starting timestamp, we vary the length of the removed block and we compute the cumulative MSE of each block. The x-axis represents the length (number of values) of the removed block to recover and the y-axis represents the average cumulative MSE. The experiments in Figures 11(a) and 11(b) are executed respectively on six different temperature time series with 1000 values each measured in region of Alto Adige and four different humidity time series with 1000 values each measured in the region of Vipetino. For these two experiments, we remove a block from one time series only while the other time series are complete. The results in both exper-



(a) Average Cumulative Error for Successive Removed Blocks in One Temperature Time Series. The correlated temperature time series used in the recovery are complete.

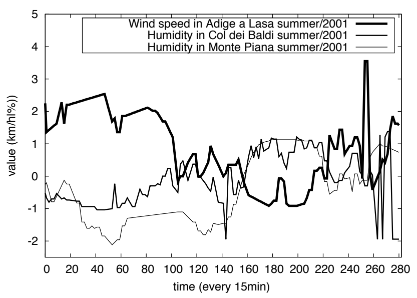


(b) Average Cumulative Error for Successive Removed Blocks in One Humidity Time Series. The correlated humidity time series used in the recovery are complete.

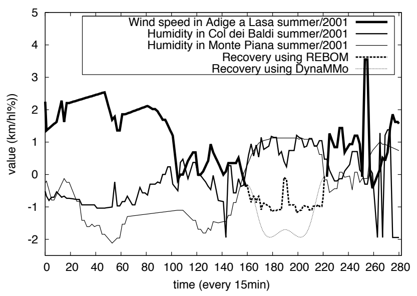


(c) Average Cumulative Error for Successive Removed Blocks in One Humidity Time Series. The correlated humidity time series used in the recovery contain missing values.

Figure 11: Recovery of Blocks of Different Lengths



(a) Time Series Measured in Three Different Areas



(b) Recovery of a Removed Block

Figure 10: Recovery Using REBOM and DynaMMo

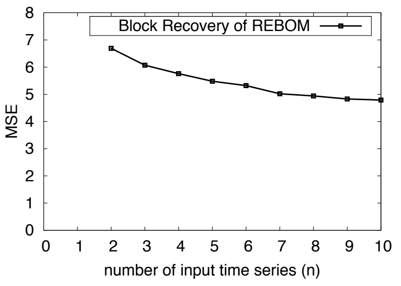
iments show that REBOM outperforms the low rank SVD and DynaMMo for the recovery of successive blocks of missing values and cubic spline is off the scale. For blocks of up to 100 removed values, the recovery error of REBOM slightly increases with the number of removed values. For blocks of more than 100 removed values, the error becomes almost stable and is not anymore affected by the number of removed values. In contrast, the recovery error of DynaMMo and the low rank SVD increases with the length of removed blocks. The small cumulative recovery error of REBOM is due to the use of different correlated time series at every iteration of the algorithm. The experiment of Figure 11(c) is executed on four humidity time series of 1000 values each. The first time series is complete, the second time series contains a missing block in the time range $[0, 100]$, the third time series contains a missing block in the time range $[100, 200]$ and the fourth time series contains a missing block in the time range $[200, 300]$. We execute the same process performed in the experiment of Figure 11(b) for the complete correlated humidity time series. The experiment shows that, compared to the result of Figure 11(b), the recovery accuracy of REBOM, DynaMMo and low rank SVD gets worse when using multiple time series with missing values. The recovery accuracy of REBOM is still better than the one of the other techniques.

In the experiment of Figure 12, we use different correlations and number of input time series (n) to evaluate the impact on the recovery MSE. We vary n and we compute the MSE of REBOM for the same block containing 90 missing values. Figure 12(a) shows that in the case of time series of high correlation ($1 \geq |\rho| > 0.7$), the MSE of REBOM decreases only slightly as n grows. REBOM is able to restore the missing block using a small number of highly correlated input time series. This result is explained by the fact that, for highly correlated time series, the starting top-k ranking matrix is similar to the partial ranking matrix. Therefore, the recovery of REBOM converges quickly. Figure 12(b) shows that, using more time series of moderate correlation ($0.7 \geq |\rho| > 0.4$), the MSE of REBOM decreases linearly. REBOM uses all the time series to perform the most accurate recovery. Figure 12(c) illustrates that, the MSE increases for input time series with low correlated time series ($0.4 \geq |\rho| > 0$).

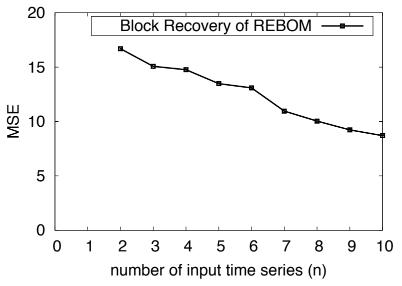
In the experiment of Figure 13 we set n to 10 and we vary the number of time series in the top-k ranking matrix. In Figure 13(a) the minimum MSE is reached for $k \in [2, 4]$. In Figures 13(b) and 13(c), the minimum recovery MSE is reached for a single value that is respectively $k = 4$ and $k = 2$. Again, the recovery accuracy of REBOM decreases for time series with low correlation, i.e., $0.4 \geq |\rho| > 0$, in the top-k ranking matrix.

5.2.2 *Invariance to Initialization Method*

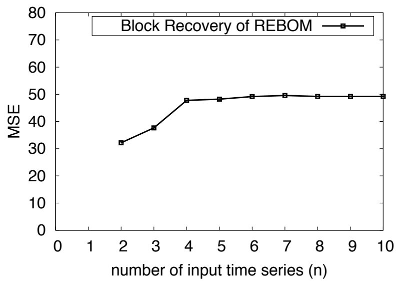
We run this experiment to test the impact of the initialization method on the block recovery of REBOM. Figure 14 shows that with different initialization techniques, REBOM needs more iterations to reach the minimum recovery error. Compared to our initialization method, a linear spline initialization needs twice the number of iterations to reach the minimum recovery error. Using a k Nearest Neighbor initialization, REBOM needs 2.5 times more iterations than



(a) Impact of Input Time Series in the Recovery MSE. Each time series has a correlation value: $1 \geq |\rho| > 0.7$

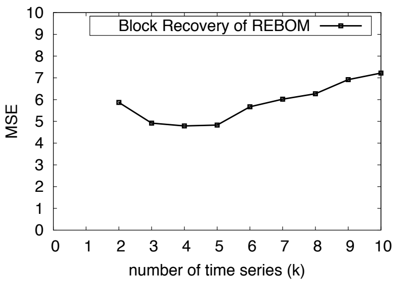


(b) Impact of Input Time Series in the Recovery MSE. Each time series has a correlation value: $0.7 \geq |\rho| > 0.4$

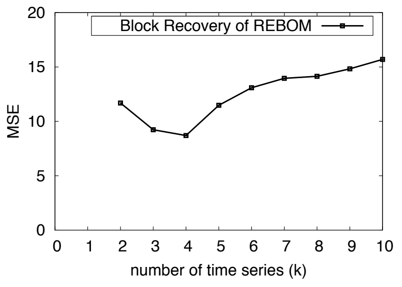


(c) Impact of Input Time Series in the Recovery MSE. Each time series has a correlation value: $0.4 \geq |\rho| > 0$

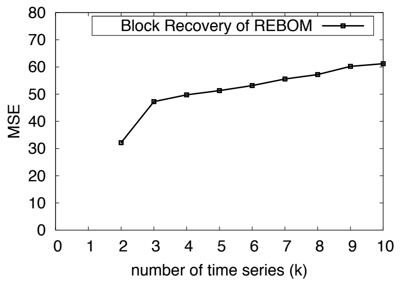
Figure 12: Impact of n in the Recovery MSE of REBOM



(a) Impact of k in the Recovery MSE. Each time series has a correlation value: $1 \geq |\rho| > 0.7$



(b) Impact of k in the Recovery MSE. Each time series has a correlation value: $0.7 \geq |\rho| > 0.4$



(c) Impact of k in the Recovery MSE. Each time series has a correlation value: $0.4 \geq |\rho| > 0$

Figure 13: Recovery MSE using different number of time series in the top-k ranking matrix

our initialization technique to reach the same recovery error. Thus, the accuracy of REBOM is independent from the initialization method. However, our initialization initialization method provides a faster recovery of blocks of missing values.

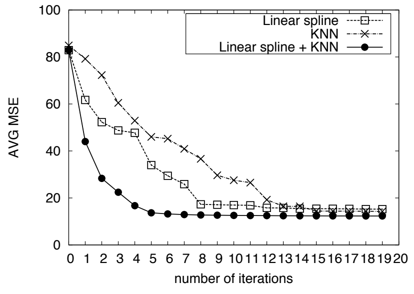


Figure 14: Number of Iterations using Different Initialization Techniques

5.2.3 *Running Time Performance*

The REBOM implementation uses the Golub/Kahan decomposition algorithm [24] and has a run time complexity of $\mathcal{O}(\#iterations \times (4n^2k + 8nk^2 + 9k^3))$, where n is the length of the longest time series and k is the number of vectors of $\mathbf{V}^{top\text{-}k}$. The complexity of building $\mathbf{V}^{top\text{-}k}$ is the cost of computing k times ρ between two time series and that is $\mathcal{O}(kn^2)$. Therefore, the total complexity of using REBOM is $\mathcal{O}(\#iterations \times (5n^2k + 8nk^2 + 9k^3))$. Figure 15 compares the total running time of REBOM against DynaMMo that has a complexity of $\mathcal{O}(\#iterations \times (kn^3))$. 3000 different time series were created by extracting 1000 observations from 15 different temperature time series.

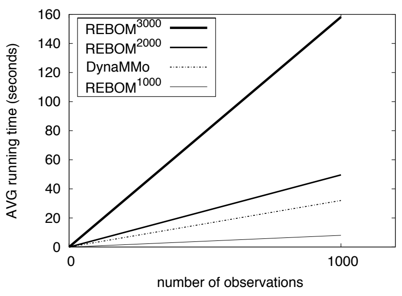
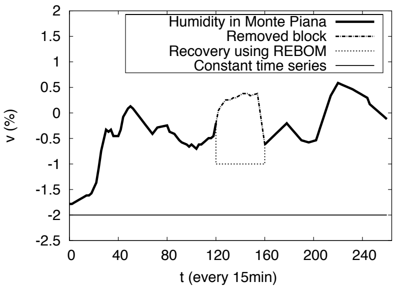


Figure 15: Average Running Time Comparison

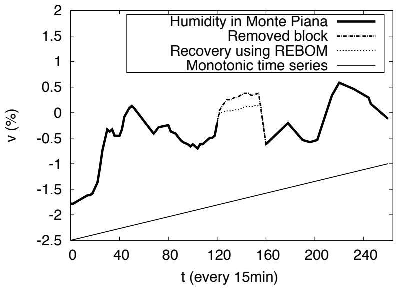
Figure 15 shows the average running time comparison performed on the created time series for the recovery of blocks containing 200 missing values. We set the value of k to four, since we reached the optimal recovery accuracy with this value. The result of this experiment shows that with 1500 time series, REBOM is faster than DynaMMo. With a higher number of input time series, the performance of REBOM starts to be slower than DynaMMo.

5.2.4 *Recovery Using Linear Time Series*

In the experiment of Figure 16, we show the impact of using extremely irregular time series. We take as input a humidity time series measured in spring 2001 from which we remove a block for $t \in]120, 160[$, a constant time series, and a monotonic time series. The result of Figure 16(a) shows that, since the correlation between the humidity time series and the constant time series is undefined (all values are equal), REBOM performs a bad recovery. In Figure 16(b), the humidity time series and the monotonic time series are correlated. Therefore, both time series are used to recover the type of the missing block. The recovered block has an increasing monotonic shape that looks similar to the monotonic time series. In the experiment of Figure 16(b), both the type and the shape of the missing block are accurately recovered. The application of DynaMMo in the experiment would set all the recovered values to 0.



(a) Recovery of REBOM using lines of function v=c, where c is a constant. The result of the recovery is the same for any given value of c



(b) Recovery of REBOM using lines of function v=at+b, where a =0.5 and b=-2.5

Figure 16: Impact of Extremely Irregular Time Series in the Recovery of REBOM

5.3 Experiments with Synthetic Regular Time Series

This subsection describes a set of experiments conducted with synthetic data. We compare the block recovery of REBOM against DynaMMo.

5.3.1 *Different Amplitudes*

Figure 17 compares the recovery of the two techniques for two regular time series having different amplitudes. The first time series is a sin(t) wave and the second time series is a sine wave multiplied by a negative scaling factor, i.e., -0.25*sin(t). For $t \in]70, 110[$, we drop a block from sin(t) and we recover it using REBOM and DynaMMo. Both techniques are able to accurately recover the missing block. REBOM uses the correlation between the two time series in order to determine the shape of the missing block, i.e, a peak. The amplitude of the missing peak is determined using the amplitude of the existing peaks from sin(t). The two techniques perform an accurate recovery for any other scaling factor of the second wave.

5.3.2 *Shifted Peaks*

Figure 18 shows two regular time series shifted in time, i.e., sin(t) and cos(t). For $t \in]70, 110[$, we drop a block from sin(t) and we recover it using REBOM and DynaMMo. REBOM is applied without initial alignment of the two time series. As expected, DynaMMo outperforms REBOM in recovering the missing block. DynaMMo is able to compute the periodicity model and performs a good block recovery. However, REBOM recovers a block that is only influenced by the shape of the block in cos(t) for $t \in]70, 110[$, i.e., a peak followed by a valley. For shifted time series, REBOM is not able to use the history of sin(t) in the recovery process. The decomposition performed by our technique is sensitive to the row position of values inside the $\mathbf{V}^{top\text{-}k}$ matrix. In order to overcome this problem, an initial alignment between the two time series must be performed in a preprocessing step (cf. Subsection 3.2).

6. CONCLUSION

This paper studies the recovery of blocks of missing values in irregular time series. We develop an iterative greedy algorithm called REBOM, that uses at every iteration the most correlated time series to the time series that contains the missing blocks to reconstruct missing peaks and valleys. Empirical studies on real hydrological data sets demonstrate that our algorithm has the most accurate block recovery among existing techniques. In future work, it is of interest to examine the impact of using the recovered time series in the recovery process instead of the original ones. It is also foreseen to investigate the impact the global correlation on the recovery accuracy together with the local correlation. Another promising direction, is to progress the interaction with the database and develop an SQL based recovery solution that reduces the number of I/O's.

Acknowledgments

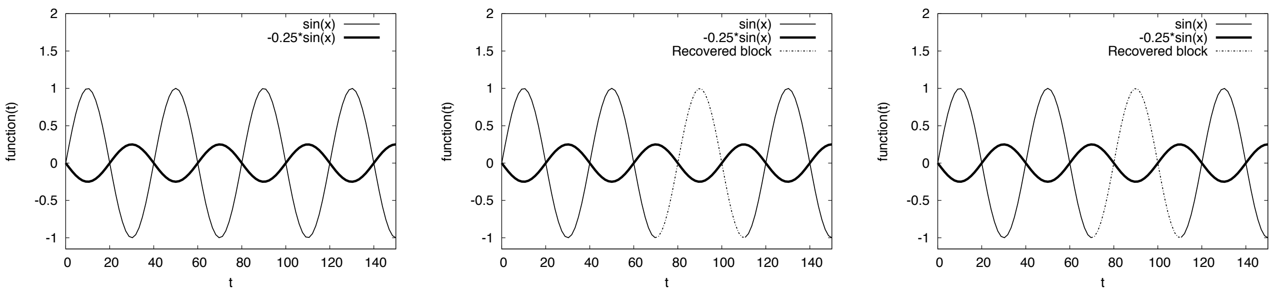
The authors would like to thank HydroloGis company [23] for providing the hydrological datasets that we have used for our experiments, and Michal Koltonik for his contribution in the implementation of REBOM. We wish also to give special thanks to the anonymous reviewers for their insightful comments.

7. REFERENCES

[1] Mueen, A., Nath. S., and Liu, J., : *Fast Approximate Correlation for Massive Time-series Data*, in SIGMOD, 2010

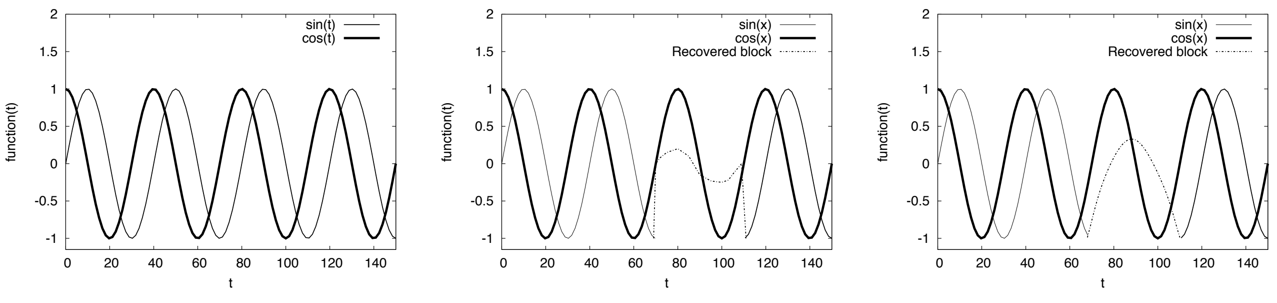
[2] Srebro, N., and Jaakkola, T., : *Weighted Low-Rank Approximations*, in ICML, 2003

[3] Tsechansky, M.S., and Provost, F., : *Handling Missing Values when Applying Classification Models*, in JMLR, 2007



(a) Original Waves (b) Recovery of REBOM (c) Recovery of DynaMMO

Figure 17: Recovery of DynaMMO and REBOM for Time Series of Different Amplitudes



(a) Original Waves (b) Recovery of REBOM (c) Recovery of DynaMMO

Figure 18: Recovery of DynaMMO and REBOM for Shifted Time Series

[4] Romero, V., and Salmerón, F., : *Multivariate Imputation of Qualitative Missing Data using Bayesian Networks*, in SMPS, 2004

[5] Harvey, M., Carman, M.J., Ruthven, I., and Crestani, F., : *Bayesian Latent Variable Models for Collaborative Item Rating Prediction*, in CIKM, 2011

[6] Srebro, N., and Jaakola, T., : *Weighted Low-Rank Approximations*, in ICML, 2003

[7] Li, L., MacCann, J., Pollard, N., and Faloutsos, C., : *DynaMMo: Mining and Summarization of Coevolving Sequences with Missing Values*, in KDD, 2009

[8] Jain, A., Chang, E.Y., and Wang, Y.F., : *Adaptive Stream Resource Management using Kalman Filters*, in SIGMOD, 2004

[9] Ding, H., et al. : *Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures*, in PVLDB, 2008

[10] Chen, Q., Chen, L., Lian, X., and Yu, J.X., : *Indexable PLA for Efficient Similarity Search*, in VLDB, 2007

[11] Gelaman, A., : *Data Analysis using Regression and Multilevel/Hierarchical Models*, Publisher: Cambridge University Press, 1 edition, 2006

[12] Yi., B.K., Sidiropoulos, N.D., Johnson, T. , Jagadish, H.V., Faloutsos, C., and Biliris, A., : *Online Data Mining for Co-Evolving Time Sequences*, in ICDE, 2000

[13] Troyanskaya, O., et al : *Optimal Multi-step k-Nearest Neighbor Search*, in J. Bioinformatics, 2001

[14] Seidl, T., and Kriegel, H.P., : *Optimal Multi-step k-Nearest Neighbor Search*, in SIGMOD, 1998

[15] Kurucz, M., Benczur, A.A., and Csalogany, K., : *Methods for Large Scale SVD with Missing Values*, in KDD, 2007

[16] Meyer, C.D., : *Matrix Analysis and Applied Linear Algebra*, Publisher: SIAM-Society for Industrial and Applied Mathematics, pages 412-417 and 489-504, 2000

[17] Kalman, D., : *A Singularly Valuable Decomposition: The SVD of a Matrix*, in J. College Mathematics, 1996

[18] Brand, M., : *Incremental Singular Value Decomposition of Uncertain Data with Missing Values*, in ECCV, 2002

[19] Alter, O., and al., : *Singular value decomposition for genome-wide expression data processing and modeling*, in PNAS, 2000

[20] Mohan, M., Chen., Z., and Weinberger, K., : *Web-Search Ranking with Initialized Gradient Boosted Regression Trees*, in JMLR, 2011

[21] He, Y., : *Missing Data Imputation for Tree-Based Models*, PhD dissertation, 2006

[22] Ding, Y., and Simonoff., J.S., : *An Investigation of Missing Data Methods for Classification Trees Applied to Binary Response Data*, in JMLR, 2010

[23] HydroloGIS company, available for online access at: `http://www.hydrologis.eu/`

[24] Erricos, J., : *Handbook on Parallel Computing and Statistics*, Book, Chapter 4, 2005

[25] Lagarias, J.C., : *Monotonicity Properties of the Toda Flow, the QR-Flow, and Subspace Iteration*, in SIAM J. Matrix Analysis and Applications, 1991

APPENDIX

A. FUNCTION COMPUTING MISSING TIME STAMPS

We consider two relations:

- *Observation* (series_id, ts, val) that stores the values of observations, where series_id is the id of time series, ts and val are respectively the time stamp and value of observations
- *Series* (id, granul) that stores information about time series, where id is the id of time series and granul is the granularity of time series, i.e., a time series has a granularity of two if the observations occur every two minutes.

Given these two relations, we define function *load_mv_indexes()* that efficiently finds the indexes of all missing time stamps. This function uses the granularity of each time series in order to create a sequence of incremental granularities, e.g., {2,4,6,...}. Then, the set difference between the sequence of granularities and the existing time stamps gives the indexes of missing time stamps. *load_mv_indexes()* is executed as an SQL function on the database server side.

```
FUNCTION load_mv_indexes (in_series_id IN
                            INTEGER) AS
    ts_lst  INTEGER;
    gran    INTEGER;
    out_mv_indexes   INTEGER;

  BEGIN
    SELECT granul
    INTO gran
    FROM Series
    WHERE id=in_series_id;

    SELECT MAX(ts)
    INTO ts_lst
    FROM Observation
    WHERE series_id=in_series_id;

    SELECT ts
    BULK COLLECT INTO out_mv_indexes
    FROM (
      SELECT * FROM (
        SELECT (level-1)*gran ts
        FROM dual
        CONNECT BY LEVEL <= (ts_lst+gran)/gran
      MINUS
        SELECT ts
        FROM Observation
        WHERE series_id=in_series_id
      ) ORDER BY 1
    );
  RETURN(out_mv_indexes);
  END;
```

B. PROOF SKETCHES

B.1 Lemma 1

Proof. We prove that our algorithm is stepwise monotonic. We perform this proof by showing that the correlation matrix used is monotonic at every step of the algorithm. i) From Def. 2 (SVD) we know that the singular values define the variances along the vectors. ii) From the definition of the top-k ranking matrix we know that at every step of SVD, we take the matrix with the biggest l-1 norm of correlation. iii) From the definition of $SVD_r()$ we know that only the smallest variance will be nullified and the biggest ones will be kept. Using i), ii) and iii) we can deduce that our algorithm takes the biggest $||\rho_{\mathbf{V}_i}||$ in order to compute the biggest $\sum_{V_{i_j} \in \mathbf{W}_i} var(V_{i_j})$ where $\mathbf{W}_i = SVD_r(\mathbf{V}_i)$. Therefore, the bigger the correlation is, the bigger sum of variances we will obtain. This implies that the correlation used by the algorithm is stepwise monotonic. $\square$

B.2 Lemma 2

Proof. We prove that our algorithm terminates after finding the matrix that has the maximum sum of variances along its vectors. We perform this proof by showing that the iterative refinement of missing values satisfies the following two properties:

- a) *finite number of rankings*: i) From Def. 2 (SVD) we know that the variance values obtained by SVD are ranked in increasing order in matrix $\mathbf{\Sigma}$. ii) From [25] we have that the singular values obtained by SVD are monotonic. Using i) and ii) it follows that the variance obtained by the decomposition is monotonic and thus: $W_{1_j} \in \mathbf{W}_1^{top\text{-}k} \wedge W_{2_j} \notin \mathbf{W}_2^{top\text{-}k} \Rightarrow W_{3_j} \notin \mathbf{W}_3^{top\text{-}k}$ where $\mathbf{W}_2^{top\text{-}k} = SVD_r(\mathbf{W}_1^{top\text{-}k})$ and $\mathbf{W}_3^{top\text{-}k} = SVD_r(\mathbf{W}_2^{top\text{-}k})$. Therefore, the number of rankings generated by our algorithm is finite and this property is satisfied.
- b) *ranking of a matrix determine the result of $SVD_r()$*: Let R_i be the ranking of matrix $\mathbf{W}_i$, $\widetilde{R}_i$ be the partial ranking of matrix $\mathbf{W}_i$ and R_{i+1} be the ranking of matrix $\mathbf{W}_{i+1}$ where $\mathbf{W}_{i+1} = SVD_r(\mathbf{W}_i)$. i) We have from Def. 3 that the correlation value determines the ranking inside a matrix and then $||\rho_{\mathbf{W}_i}|| = ||\rho_{\mathbf{W}_{i+1}}|| \Rightarrow R_i = R_{i+1}$. ii) Since UMV algorithm (cf. Subsection 4.4) is updating only the missing values of the matrix, then: $\widetilde{R}_i$ determines $||\rho_{\mathbf{W}_{i+1}}||$ and it follows that: $R_i = \widetilde{R}_i \Rightarrow ||\rho_{\mathbf{W}_i}|| = ||\rho_{\mathbf{W}_{i+1}}||$. Using i) and ii) we deduce by transitivity that: $R_i = \widetilde{R}_i \Rightarrow R_i = R_{i+1}$ and therefore, this property is satisfied.

Properties a) and b) hold for matrices whose vectors are correlated. It follows the proof for this lemma. $\square$

A Novel Query-Based Approach for Addressing Summarizability Issues in XOLAP

Marouane Hachicha Chantola Kit Jérôme Darmont

Université de Lyon (ERIC Lyon 2)
5 avenue Pierre Mendès-France
69676 Bron Cedex
France
marouane.hachicha@univ-lyon2.fr, kchantola@gmail.com, jerome.darmont@univ-lyon2.fr

ABSTRACT

The business intelligence and decision-support systems used in many application domains casually rely on data warehouses, which are decision-oriented data repositories modeled as multidimensional (MD) structures. MD structures help navigate data through hierarchical levels of detail. In many real-world situations, hierarchies in MD models are complex, which causes data aggregation issues, collectively known as the summarizability problem. This problem leads to incorrect analyses and critically affects decision making. To enforce summarizability, existing approaches alter either MD models or data, and must be applied *a priori*, on a case-by-case basis, by an expert. To alter neither models nor data, a few query-time approaches have been proposed recently, but they only detect summarizability issues without solving them. Thus, we propose in this paper a novel approach that automatically detects and processes summarizability issues at query time, without requiring any particular expertise from the user. Moreover, while most existing approaches are based on the relational model, our approach focus on an XML MD model, since XML data is customarily used to represent business data and its format better copes with complex hierarchies than the relational model. Finally, our experiments show that our method is likely to scale better than a reference approach for addressing the summarizability problem in the MD context.

1. INTRODUCTION

Business intelligence and decision-support systems in general are nowadays used in many business (e.g., finance, telecoms, insurance, logistics) and non-business (e.g., agriculture, medicine, health and environment) domains. Such systems casually rely on data warehouses, which are designed, both at the conceptual and logical levels, using multidimensional (MD) structures [28]. In MD models, facts are analysis subjects of interest (e.g., sales) that are described by a set of (usually numerical) measures (e.g., sale quantity and amount) w.r.t. analysis axes called dimensions (e.g., book category, sale date, sale location...). Dimensions may be organized in hierarchical levels to allow data aggregation at different granularities (e.g., store, city, state or country, from the finer level to the coarser level).

MD modeling essentially aims at easing online analytical processing (OLAP), whose main operators help navigate data through coarser (roll up) and finer (drill down) levels of detail. In this context, aggregating measures works fine when intradimensional relationships are one-to-many (e.g., a book belongs to one single category). However, in real-world situations, dimension hierarchies may be much more complex [3, 17], which leads to a semantic gap between MD models and current OLAP tools [28], an issue known as the summarizability problem [13]. Violating summarizability is a critical matter, for it causes erroneous aggregations and, therefore, erroneous analyses that can jeopardize important decisions [21]. However, testing summarizability is a difficult (coNP-complete) problem [10]. Finally, complex hierarchies are difficult to both represent in classical database management systems and query with SQL-like languages, while XML storage and interrogation with XQuery is much more natural [3], which led to the design of XML data warehouses and so-called XOLAP solutions.

The summarizability problem is widely acknowledged as crucial and has received some attention in the Nineties, with most solutions aiming at *a priori* normalizing data to enforce summarizability. Quite surprisingly, few researchers came back on this topic since then, although we identify two types of shortcomings in normalization approaches. First, normalizing data breaks initial conceptual MD models, provoking the alteration or loss of some semantics. Thus, there would be no point in exploiting XML's flexibility to model rich, complex hierarchies if they were "flattened" after normalization. Second, data normalization applies *a priori*, on a case-by-case basis, and requires the intervention of an expert in MD modeling. Such an approach is subjective, likely to be costly and does not scale well w.r.t. data volume [20]. Finally, to the best of our knowledge, there is no existing XOLAP approach that provides a practical solution to summarizability issues, while they are much likely to occur in an XML data warehouse with complex dimension hierarchies. The closest approach does detect summarizability issues, but then returns no result [22, 23].

Thus, we propose in this paper a novel approach, set in the

The 18th International Conference on Management of Data (COMAD), 14th-16th Dec, 2012 at Pune, India.
Copyright ©2012 Computer Society of India (CSI).

XOLAP context, to the summarizability problem. By contrast to normalization, our approach does not alter data to retain all semantics. We also favor paying the price of some overhead and tackle the summarizability problem at query time, without requiring any expertise beyond the user's, to avoid re-normalizing when data schema evolves, favor scalability and eliminate human-related costs. In many institutions, decision-support applications indeed require external Web data [7]. Due to the heterogeneity and high evolutivity of such data, an XOLAP run-time solution is more suitable than *a priori* expert interventions.

The remainder of this paper is organized as follows. In Section 2, we formalize the background information related to data warehouses, and define what we term complex hierarchies and summarizability. We also review the existing approaches for enforcing summarizability. In Section 3, we motivate and introduce our query-based solution to complex hierarchy management in XOLAP, including novel pattern tree-based data and query models, as well as the aggregation algorithm that exploits them. In Section 4, we provide a complexity study and an experimental validation of our work. Finally, in Section 5, we conclude this paper and hint at future research.

2. BACKGROUND

In this section, we formalize data warehousing concepts and define complex hierarchies that lead to summarizability issues. Then, we discuss the approaches that address the summarizability problem.

2.1 Data Warehouses

2.1.1 Data Warehouse

A data warehouse W modeled w.r.t. a snowflake schema (i.e., with dimension hierarchies) is defined as $W = (\mathcal{F}, \mathcal{D})$, where $\mathcal{F}$ is a set of facts to observe and $\mathcal{D}$ is a set of dimensions or analysis axes. Let $d = |\mathcal{D}|$.

2.1.2 Dimension and Hierarchy

$\forall i \in [1, d]$, a dimension $D_i \in \mathcal{D}$ is defined as a hierarchy made up of a set of n_i levels: $D_i = \{\mathcal{H}_{ij} | j = 1, n_i\}$. By convention, we denote $\mathcal{H}_{i1}$ as the lowest granularity level. $\forall j \in [1, n_i]$, a hierarchy level $\mathcal{H}_{ij}$ is defined in intention as $\mathcal{H}_{ij} = (ID_{ij}, \{A_{ijk} | k = 1, a_{ij}\}, R_{ij})$, where ID_{ij} is the identifier attribute of $\mathcal{H}_{ij}$, $\{A_{ijk}\}$ is a set of a_{ij} so-called member attributes of $\mathcal{H}_{ij}$, and R_{ij} is an attribute that references a hierarchy level at a higher granularity than that of $\mathcal{H}_{ij}$ (notion of roll up).

Let $dom()$ be a function that associates to any attribute its definition domain. Let $h_{ij} = |\mathcal{H}_{ij}|$. $\forall l \in [1, h_{ij}]$, instances of $\mathcal{H}_{ij}$ are tuples $H_{ijl} = (\sigma_{ijl}, \{\alpha_{ijkl} | k = 1, a_{ij}\}, \rho_{ijl})$, where $\sigma_{ijl} \in dom(ID_{ij})$, $\alpha_{ijkl} \in dom(A_{ijk})$ $\forall k \in [1, a_{ij}]$, and $\rho_{ijl} \in dom(ID_{ij'})$ with $j' \in [1, n_i]$.

2.1.3 Fact

The set of facts $\mathcal{F}$ is defined in intention as $\mathcal{F} = (\{\Delta_i | i = 1, d\}, \{M_j | j = 1, m\})$, where $\{\Delta_i\}$ is a set of d attributes that reference instances of hierarchy levels $\mathcal{H}_{i1}$ of each dimension $D_i \in \mathcal{D}$, and $\{M_j\}$ is a set of m measure (or indicator) attributes that characterize facts.

Let $f = |\mathcal{F}|$. $\forall k \in [1, f]$, instances of $\mathcal{F}$ are tuples $F_k = (\{\delta_{ik} | i = 1, d\}, \{\mu_{jk} | j = 1, m\})$, where $\delta_{ik} \in dom(ID_{i1})$ $\forall i \in [1, d]$, and $\mu_{jk} \in dom(M_j)$ $\forall j \in [1, m]$.

2.2 Complex Hierarchies

We term a dimension hierarchy D_i as complex if it is both non-strict and incomplete. We choose this new, general denomination because dimension hierarchy characterizations vary wildly in the literature. For example, Beyer et al. name complex hierarchies ragged hierarchies [3], while Rizzi defines ragged hierarchies as incomplete only [27]. Malinowski and Zimányi also use the terms of complex generalized hierarchy [17], but even though they include incomplete hierarchies, they do not include non-strict hierarchies.

2.2.1 Non-Strict Hierarchy

A hierarchy is non-strict [1, 16, 30] or multiple-arc [27] when attribute R_{ij} is multivalued. In other terms, from a conceptual point of view, a hierarchy is non-strict if the relationship between two hierarchical levels is many-to-many instead of one-to-many. For example, in a dimension describing products, a product may belong to several categories instead of just one.

Similarly, a many-to-many relationship between facts and dimension instances may exist [27]. For instance, in a sale data warehouse, a fact may be related to a combination of promotional offers rather than just one. Formally, here, attributes Δ_i ($\forall i \in [1, d]$) may be multivalued.

2.2.2 Incomplete Hierarchy

A hierarchy is incomplete [4, 25], non-covering [1, 16, 30] or ragged [27] if attribute R_{ij} allows linking a hierarchy level $\mathcal{H}_{ij}$ to another hierarchy level $\mathcal{H}_{ij'}$ by "skipping" one or more intermediary levels, i.e., R_{ij} refers to $ID_{ij'}$ such that $j' > j + 1$. This occurs, for instance, if in a dimension describing stores, the store-city-state-country hierarchy allows a store to be located in a given region without being related to a city (stores in rural areas).

Similarly, facts may be described at heterogeneous granularity levels. For example, still in our sale data warehouse, sale volume may be known at the store level in one part of the world (e.g., Europe), but only at a more aggregate level (e.g., country) in other geographical areas. This means that $\forall i \in [1, d]$, $\delta_i \in dom(ID_{ij})$ with $j \in [1, n_i]$ (constraint $j = 1$ is forsaken).

A particular case of incomplete hierarchies are called non-onto [24], heterogeneous [10], unbalanced [9, 17] or asymmetric [16] hierarchies. A hierarchy is non-onto when all paths from the root to a leaf in the hierarchy do not have equal lengths [24], but here, missing elements are always child nodes, while they may be parent nodes in an incomplete hierarchy.

Note that some papers addressing the summarizability problem differentiate between intradimensional relationships and fact-to-dimension relationships [20]. By contrast, as Pedersen et al. [24], we consider that summarizability issues and solutions are the same in both cases, since facts may be viewed as the very finer granularity in the dimension set.

2.3 Summarizability in MD Models

The notion of summarizability was introduced by Rafanelli and Shoshani in the context of statistical databases [26], where it refers to the correct computation of aggregate values with a coarser level of detail from aggregate values with a finer level of detail. Then, Lenz and Shoshani defined three constraints that guarantee summarizability in the MD context [13]: (1) hierarchies must be strict; (2) hierarchies must be complete; (3) aggregate data types must be compatible, i.e., an aggregate function must be applicable to a given measure for a given set of dimensions. For instance, a maximum sale amount is a meaningful aggregation, while a sum of temperatures would be meaningless. These constraints also hold for fact-to-dimension relationships [20]. In this paper, we assume that the type compatibility constraint is handled by users.

One way to ensure summarizability in a MD model is to simply disallow complex hierarchies at design time, as in the Dimensional Fact Model [5]. However, to support different kinds of complex real-world situations, most MD models do allow complex hierarchies. Thence, the summarizability problem must be addressed. There are two main families of approaches: schema normalization and data transformation, which are reviewed below. Both families of approaches operate at design time.

More recent proposals operate at query time, but they are very few. Guidelines have been proposed for tolerating and displaying incorrect aggregation results [8], but they have not been implemented. The generalized projection XOLAP operator [22, 23] detects summarizability issues, but does not solve them and returns an error flag instead.

Finally, the interested reader may find more details about summarizability issues in the survey by Mazón et al. [21].

2.3.1 *Schema Normalization*

Two strategies may be used to achieve schema normalization. The first strategy is based on the definition of constraints and transformation rules. For instance, Hurtado et al. propose a class of integrity constraints to address incompleteness, namely dimension constraints and frozen dimensions [10]. Frozen dimensions are minimal, complete dimensions mixed up in incomplete dimensions using dimension constraints that help model incomplete hierarchy schemas. From their part, Lechtenbörger and Vossen introduce new MD normal forms (MNFs) [12]. 1MNF does not allow non-strict hierarchies, while 2MNF and 3MNF permit to model incomplete relationships using context dependencies, i.e., dimension constraints. Specialization constructs in dimensions can lead to incomplete relationships [13, 26] and context dependencies enable an implicit representation of such specializations.

The second strategy adds new structures into the model in order to ensure summarizability. In relational implementations, bridge tables are used to capture non-strict fact-to-dimension relationships via foreign keys that refer to the dimension and fact tables [11, 29]. Arguing that bridge tables defined at the logical level make the modeling of complex structures difficult, some authors introduce their equivalent at the conceptual level [16, 20]. Such additional entities/classes help store instances at the origin of incompleteness and/or non-strictness. Finally, Mansmann and Scholl propose a two-phase modeling approach that transform incomplete hierarchies into a set of well-behaved subhierarchies without summarizability problems [18, 19].

2.3.2 *Data Transformation*

The reference data transformation approach by Pedersen et al. transforms dimension and fact instances to enforce summarizability [24]. To solve incompleteness, all mappings between hierarchical levels are transformed to be complete with the help of an algorithm named `MakeCovering`. For example, suppose that some addresses are missing in an address-city-country hierarchy. `MakeCovering` inserts new values into the missing hierarchical level address to ensure that mappings to higher hierarchical levels are summarizable. `MakeCovering` exploits metadata and/or expert advice for this sake. For example, an expert would be required to recover missing addresses in small streets in the USA or Australia. The authors also propose a simplified version of `MakeCovering`, `MakeOnto`, to handle summarizability in non-onto hierarchies by replacing childless nodes by so-called placeholder values.

Mappings are made strict with the help of another algorithm named `MakeStrict`. `MakeStrict` avoids “double counting” by “fusing” multiple values in a parent hierarchical level into one “fused” value, and then linking the child value to the fused value. Fused values are inserted into a new hierarchical level in-between the child and the parent. Reusing this new level for computing higher-level aggregate values leads to correct aggregation results.

Mansmann and Scholl further modify Pedersen et al.'s algorithms to eliminate roll up/drill down incomplete and non-strict hierarchies at the instance level [18, 19]. Finally, Li et al. demonstrate that `MakeCovering` does not work on some real-world cases, i.e., geographical hierarchies in China [14]. They identify four types of incompleteness that are specific to China and thence propose several variations of `MakeCovering` to handle them.

3. QUERY-BASED COMPLEX HIERARCHY MANAGEMENT IN XOLAP

3.1 Motivation and Contributions

In XML data warehouses and XOLAP, complex data structures, and especially complex hierarchies, are likely to be present, and are likely to evolve with time faster than in legacy decision-support systems. In such a context, summarizability cannot be enforced through a costly [20] data normalization process each time schema and data are updated. Thus, as in the most recent existing approaches [8, 22, 23], we advocate for a run-time solution.

However, while existing run-time approaches do detect summarizability issues and warn the user, they still output incorrect or absent results. Our first contribution is thus to complete the process and output correct results. To achieve this goal, we adapt and automatize well-known solutions from the literature (Sections 3.2 and 3.4). Since we operate at query time, we deliberately adopt simple and robust solutions not to add too much overhead over summarizability testing. Such reference approaches are still customarily reused and adapted by recent approaches [18, 19].

Furthermore, all XOLAP approaches we are aware of propose operators under the form of ad-hoc programs, and rely on relational database systems, including Pedersen et *al.*'s [22, 23]. By contrast, we aim at contributing to build an XOLAP algebra that can later translate into standard XQuery statements. Thus, our second contribution introduces data and query models based on the data trees and tree patterns used in XML processing [6], respectively (Section 3.3).

3.2 Principle of our Approach

To illustrate how our approach operates, let us consider the example from Figure 1, which represents a complex "project management" hierarchy at the instance level, adapted from [18, 19]. This hierarchy is non-strict because teams may manage several projects (Team 2 manages projects A and B), while projects may be managed by several teams (projects A and B are managed by teams 1 and 2, and teams 2 and 3, respectively). The hierarchy is also incomplete, since Project D is not managed by any particular team; thus it is complex.

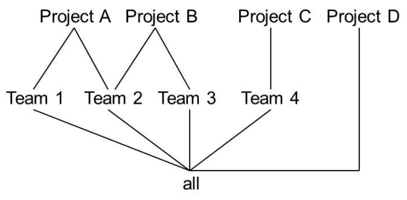


Figure 1: Sample complex hierarchy

First, to handle non-strict hierarchies in a given dimension D_i, we must avoid multiplying the aggregation of instance measures of a hierarchy level $\mathcal{H}_{ij}$ when rolling up to level $\mathcal{H}_{ij+1}$. Thus, when building the set of groups G with respect to a grouping criterion, we fuse multiple values in $\mathcal{H}_{ij+1}$ into one single "fused value", i.e., we build $G = \bigcup_{l \in [1,h_{ij}]} \rho_{ijl}$, where multivariate values of ρ_{ijl} are considered as sets instead of single values. In our example, suppose we are counting projects per teams for projects A and B. Then $G_{NS} = \{\{Team\ 1, Team\ 2\}, \{Team\ 2, Team\ 3\}\}$. The number of projects in $\{Team\ 1, Team\ 2\}$ is 1, the number of projects in $\{Team\ 2, Team\ 3\}$ is 1, for a correct total of 2. If G_{NS} had been $\{Team\ 1, Team\ 2, Team\ 3\}$, the total number of projects would have been wrong $(1 + 2 + 1 = 4)$ in $\mathcal{H}_{12}$.

Second, to handle incomplete hierarchies, we must, when rolling up from a hierarchy level $\mathcal{H}_{ij}$ to level $\mathcal{H}_{ij+1}$, still aggregate measures of instances of $\mathcal{H}_{ij}$ that are *not* present in $\mathcal{H}_{ij+1}$. Thus, when building G, all "missing instances" are grouped into an artificial "Other" group, i.e., $G = \bigcup_{l \in [1,h_{ij}]} \rho_{ijl} \cup \{Other\}$ such that $\nexists l'/\rho_{ijl} = \sigma_{i(j+1)l'}$. In our example, suppose we are again counting projects per teams, but for projects C and D. Then $G_I = \{Team\ 4, Other\}$. The number of projects in G_I is 2, whereas it would have been wrong, i.e., 1, if G_I had been $\{Team\ 4\}$ only.

Third, to handle complex hierarchies, we simply apply both the managements of non-strict and incomplete hierarchies. Thus, here, $G = \bigcup_{l \in [1,h_{ij}]} \rho_{ijl} \cup \{Other\}$ such that $\exists L/\rho_{ijl} = \bigcup_{l' \in L} \sigma_{i(j+1)l'}$. In our example, if we are now counting projects per teams for all projects, then $G_C = G_{NS} \cup G_I$, and the number of projects in G_C is correct, i.e., 4.

Finally, note that, beyond the expert-based preprocessing vs. our automatic, on-the-fly approach, there is a substantial difference between our view of incomplete hierarchy management and Pedersen et *al.*'s reference solution [24]. While they call to an expert to replace all "missing values" in G by actual values, we indeed automatically add an "Other" group for all "missing values" of a given hierarchical level. "Other" values from different hierarchy levels are of course distinguished, e.g., `Project[Other]` is different from `Team[Other]`.

Thus, we presumably loose in semantical finesse, but we spare the cost of the expert. Moreover, the simplicity of our approach helps handle all cases of incompleteness identified by Li et al. [14], while `MakeCovering` cannot.

3.3 Data and Query Models

3.3.1 Data Model

Since complex hierarchies have been shown to be better represented in XML at the physical level [3], we choose XML to model MD data. Thus, at the logical level, we choose XML data trees to model MD structures. Data trees are indeed casually used to represent and manipulate XML documents, whose hierarchical structure is akin to a labeled ordered, rooted tree [6]. Moreover, data trees allow modeling MD structures. Formally, a data tree t models an XML document or a document fragment. It can be defined as a triple $t = (r, N, E)$, where N is the set of nodes, $r \in N$ is the root of t, and E is the set of edges stitching together couples of nodes $(n_i, n_j) \in N \times N$.

Figure 2 shows how we logically model MD data with a data tree. $*$-labeled edges indicate a one-to-many relationship. The data tree root, W, models the data warehouse. Its child nodes F model facts. Each fact is described by a set of dimensions D and measures M. For a given fact, we may have several dimensions (such as client, supplier...) and several measures (such as account, quantity...). A dimension hierarchy can have any number of levels H. The $*$ multiplicity on the D-H edge allows facts to roll up to any number of hierarchy levels, at any granularity (fact-to-dimension relationships). The recursive edge on H allows any hierarchy level to roll up to several higher levels, possibly skipping any number of intermediary levels (intradimension relationships). Thus, this representation permits to model complex hierarchies.

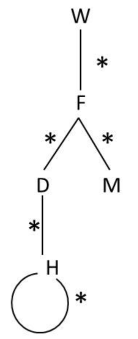


Figure 2: Multidimensional data tree model

Figure 3 exemplifies the instantiation of our model by elaborating on the complex hierarchy from Figure 1. Here, facts are described by a project and a customer dimension, and the only measure is cost.

3.3.2 Query Model

Since we use XML data trees as our logical data model, we use XML tree patterns, which are the most efficient structures to query data trees [6], as our query model. A tree pattern (TP) or tree pattern query is a pair (t, F) where t is a data tree (r, N, E). An edge in t may either be a

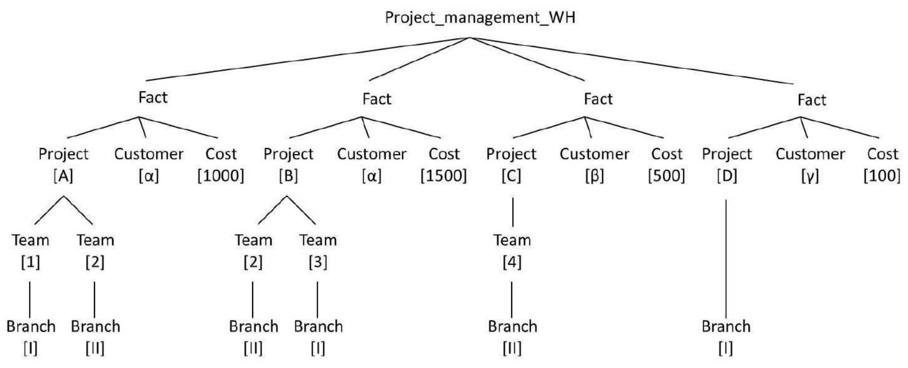


Figure 3: Sample multidimensional data tree

parent-child (*pc* for short, simple edge in XPath) node relationship or an ancestor-descendant (*ad* for short, double edge in XPath) node relationship. F is a formula that specifies constraints on TP nodes. More explicitly, F is a boolean combination of predicates on TP node values. For example, the TP from Figure 4(a) selects all projects whose cost is strictly greater than 1000. Matching this TP against the data tree from Figure 3 outputs a new data tree, also called witness tree (WT), which is depicted in Figure 4(b).

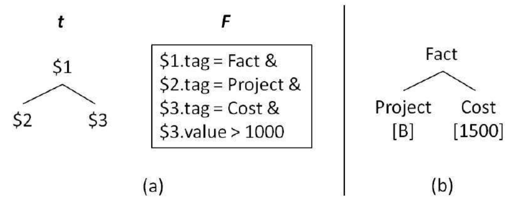
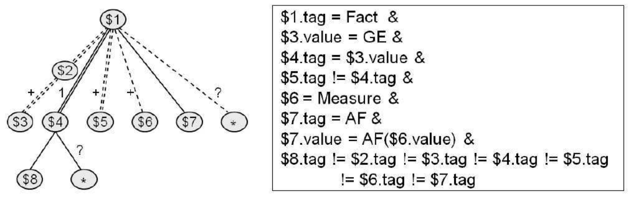


Figure 4: Sample pattern (a) and witness (b) trees

To help query MD data modeled w.r.t. Figure 2's data tree model, we propose the TP model depicted in Figure 5. In this TP model, nodes connected to their parent nodes with a dotted edge do not appear in the WT, unlike nodes connected to their parent nodes with a solid edge. Moreover, for each edge (u, v) where u is a parent (or an ancestor) of v: a "+" label means that one or many matches of v are allowed for each match of u in a WT; a "?" label means that zero to one match of v is allowed for each match of u in a WT; and a "1" label means that one and only one match of v is allowed for each match of u in a WT. Nodes from our TP model are tagged with $$i$ (i being a number) or with $*$. Nodes tagged with $*$ are always connected to their parent nodes with a *pc* relationship (/). In XPath 2.0 [2], a path $x/*$ such that x is a node returns a different result from the path $x//*$. $x/*$ returns the hierarchy connected to x while $x//*$ returns the same result as $x/*$ but with duplicate nodes. Thus, we choose to respect the XPath 2.0 standard.

Formula F precises how our TP model matches a MD data tree, as follows. Node $1 matches one fact. Node $3 specifies one to many grouping elements (denoted GE in F). A grouping element $3 is a hierarchical level. Node $2 models all nodes that may exist between $1 and $3. Node $4 receives $3's content in order to match only one node corresponding to each grouping element. Node $4's content finally receives a group G (Section 3.2). Node $5 matches all dimension hierarchical levels different from $4. These matched nodes do not appear in the WT because the corresponding dimensions do not belong to grouping elements. $6 specifies one to several measures required for aggregation purposes. Node $7 stores the result of applying an aggregation function (e.g., sum, count, etc.) AF on nodes $6. There is no guarantee that all the nodes output when matching the $*$ child nodes of $4 against a MD data tree appear in the WT due to incomplete hierarchies. Thus, node $8 retains the matching result of the $*$ child nodes of $1, except measures not used in any aggregation.



```
$1.tag = Fact &
$3.value = GE &
$4.tag = $3.value &
$5.tag != $4.tag &
$6 = Measure &
$7.tag = AF &
$7.value = AF($6.value) &
$8.tag != $2.tag != $3.tag != $4.tag != $5.tag
        != $6.tag != $7.tag
```

Figure 5: Multidimensional tree pattern model

3.4 Grouping and Roll up Algorithms

In this section, we translate the principles from Section 3.2 into a grouping algorithm called Query-Based Summarizability (`QBS`) that exploits the data and query models from Section 3.3. Then, we devise a roll up operator based on `QBS`.

`QBS` (Algorithm 1) essentially processes a "group by" query with respect to any number of grouping criteria, and additionally handles summarizability issues on the fly. `QBS` inputs: (1) a data tree D modeled w.r.t. Figure 2's data tree model and (2) a TP TPQ modeled w.r.t. Figure 5's TP model. `QBS` outputs a list of WTs $WTlist$ (i.e., a set of at least one WT). `QBS` proceeds into two main steps: (1) incompleteness and non-strictness management; (2) group matching to construct correct aggregation results.

More precisely, `QBS` first initializes $WTlist$ to empty. Then, for each fact, a variable $Group\_list$, which stores together all possible groups from different grouping elements, is also initialized to empty. Such groups are stored in the $Group$ variable, which comprises node values matched by $4 in TPQ

Algorithm 1 QBS grouping algorithm

```
 1: Input:
 2:     D       // Data tree
 3:     TPQ // Tree pattern
 4: WTlist ← ∅
 5: for all $1 do
 6:     // Step #1: Summarizability processing
 7:     Group_list ← ∅
 8:     for all $4 do
 9:         Group ← Group ∪ $4.value
10:         if $4 ∉ $1.children() and Group.nbElements() <
            $1.currentChild().nbChildren() then
11:             Group ← Group ∪ "Other"
12:         end if
13:         Group_list ← Group_list ∪ Group
14:     end for
15:     // Step #2: Group matching
16:     WT ← WTlist.exists(Group_list)
17:     if WT ≠ ∅ then
18:         WT.update($6, $7)
19:     else
20:         WT.create(D, TPQ)
21:         WTlist ← WTlist ∪ WT
22:     end if
23: end for
24: return product(WTlist)
```

(Step #1). In case of missing instances from a hierarchical level of the grouping element (if statement on line 10), the "Other" value is concatenated to *Group*. The test on line 10 means that \$4 is not a child (i.e., dimension) node of the current fact and the number of elements in *Group* is inferior to the number of edges rooted at the current dimension node (i.e., presence of an incomplete hierarchy). When a new group list is about to be built, the algorithm tests its existence in *WTlist*, i.e., it tests whether there exists a WT from *WTlist* where a node tagged with the same grouping elements has a value equal to the group list's. If true, the aggregation node is updated with current measures. Otherwise, a new WT is added to *WTlist* w.r.t. *TPQ*. Finally, all WTs are regrouped together under a unique root with the help of the `product()` function.

The description of all functions called in `QBS` follows.

- x.`children()` returns the set of child nodes of node x.
- x.`nbChildren()` returns the number of children of node x. If our context, this function returns the number of edges rooted at x.
- x.`currentChild()` returns the current child of node x.
- G.`nbElements()` computes the number of elements in group G.
- *Tlist*.`exists(`*Glist*`)` returns the data tree containing group *Glist* from one of the trees of *Tlist*, and $\emptyset$ otherwise.
- T.`update(`x, y`)` updates the value of node y from tree T with the value of node x.
- T.`create(`D, TPQ`)` creates a tree T by matching TP TPQ against data tree D.
- `product(`*Tlist*`)` regroups together all trees from tree set *Tlist* under one single root.

Eventually, a roll up operation is simply achieved by calling `QBS` several times, in sequence, with the output tree of each stage becoming the input tree of the next stage (Figure 6). For example, let us consider the MD data tree from Figure 3 and query Q1 = "total cost of projects per team and per customer", which translates into a TP whose formula is provided in Figure 7.

```
$1.tag = Fact  &
$3.value = {Team, Customer} &
$4.tag = $3.value &
$5.tag != $4.tag &
$6 = Cost &
$7.tag = Sum &
$7.value = sum($6.value) &
$8.tag != $2.tag != $3.tag != $4.tag != $5.tag
          != $6.tag != $7.tag
```

Figure 7: Q1 TP formula

For fact Project[A], `QBS` builds *Group* = 1-2. A first WT is thus created into *WTlist* w.r.t. Figure 7's TP, with dimension nodes (grouping element instances) Team[1-2] and Customer[α], and an aggregation node Sum[1000]. For fact Project[B], *Group* = 2-3 is built. Then, the algorithm checks whether there exists a WT in *WTList* containing the *Group_list* (Team[2-3], Customer[α]). As the answer is no, a second WT is created with dimension nodes Team[2-3] and Customer[α], and aggregation node Sum[1500]. Similarly, for fact Project[C], *Group* = 4 is built and a new WT is created with dimension nodes Team[4] and Customer[β], and aggregation node Sum[500].

For fact Project[D], there is no grouping element. Thus, we build *Group* = Other and a new WT is created with dimension nodes Team[Other] and Customer[γ], and aggregation node Sum[100]. Here, `QBS` traverses all elements of the hierarchy associated to Project[D] before assigning "Other" to *Group*. Finally, all created WTs in *WTlist* are appended under the same root (Figure 8). Note that the hierarchy of branches is always saved in WTs. `QBS` exploits the hierarchy schema (metadata) to consider Group[Other] as the parent element of Branch[I] in the corresponding WT.

To complete the roll up operation, i.e., aggregating on branches from the aggregation already computed on groups, `QBS` inputs a new TP corresponding to Q2 = "total cost of projects per branch and per customer", whose formula is given in Figure 9, and the result tree from Figure 8.

```
$1.tag = Fact  &
$3.value = {Branch, Customer} &
$4.tag = $3.value &
$5.tag != $4.tag &
$6 = Cost &
$7.tag = Sum &
$7.value = sum($6.value) &
$8.tag != $2.tag != $3.tag != $4.tag != $5.tag
          != $6.tag != $7.tag
```

Figure 9: Q2 TP formula

For fact Team[1-2], *Group* = I-II is built and a WT is created with dimension nodes Branch[I-II] and Customer[α], and aggregation node Sum[1000]. For fact Team[2-3], *Group* = I-II is built. Then, `QBS` checks whether *Group_list* (Branch

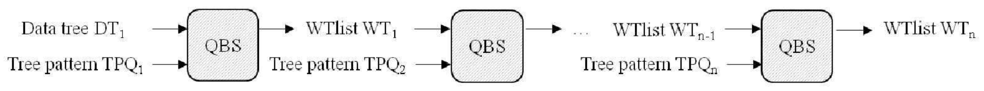


Figure 6: Roll up process

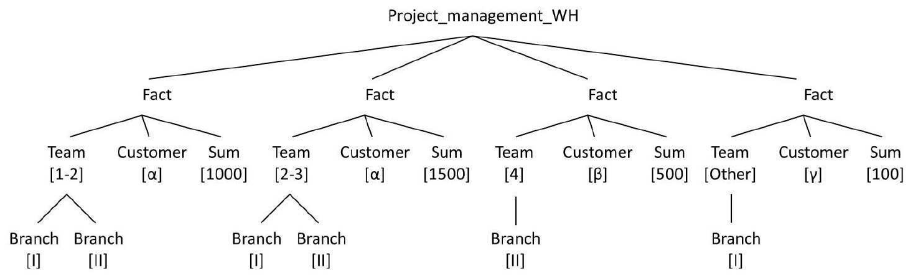


Figure 8: Sample roll up operation – Step #1

[I-II], Customer[α]) exists in *WTlist*. Here, the answer is yes, and the value of the Sum node in the returned WT is updated to 2500. For fact Group[4], *Group* = II is built. After checking the presence of *Group_list* (Branch[II], Customer[β]) in *WTlist* (which is negative), a new WT is created with dimension nodes Branch[II] and Customer[β], and aggregation node Sum[500]. For fact Team[Other], a new WT is created with dimension nodes Branch[I] and Customer[γ], and aggregation node Sum[100]. Finally, all created WTs in *WTList* are again appended under the same root (Figure 10).

4. VALIDATION

Although we should test our approach against other query-based [8], and especially XOLAP [22, 23] approaches, these approaches do detect summarizability issues, but then do not output actual aggregates. Thus, though Pedersen et al.'s reference approach [24] and its fairly recent enhancements [18, 19] apply once *a priori*, we can only compare our approach with it. Moreover, we particularly focus on the `MakeStrict` and `MakeCovering` algorithms, since `MakeCovering` generalizes `MakeOnto`. For conciseness, we label the combination of `MakeStrict` and `MakeCovering` as `Pedersen` in the following.

4.1 Complexity Study

Let us recall Section 2.1's notations: f is the number of facts in the data warehouse and d the number of dimensions. Moreover, let s be number of subdimensions, i.e., branches in non-strict hierarchies. When processing summarizability in `QBS` (Step #1 of the algorithm), for each fact and each dimension, we need to check missing values in hierarchies and replace them by "Other", and then to check whether the value exists in the current group. Thus, $f \times d \times (1 + 2 + ... + s - 1)$ tests must be performed in the worst case. Thus, the complexity of summarizability processing is $\mathcal{O}(fds^2)$.

Furthermore, when performing aggregation, for each fact, we need to check whether a group exists. Following the same reasoning, the complexity of group matching (Step #2 of `QBS`) is thus $\mathcal{O}(f^2ds^2)$. Thus, the global complexity of `QBS` is $\mathcal{O}(f^2ds^2) + \mathcal{O}(fds^2) = \mathcal{O}(f^2ds^2) \approx \mathcal{O}((fds)^2)$.

Since fds represents the input size, if we state that $n = fds$, then the worst-case complexity of `QBS` is $\mathcal{O}(n^2)$, i.e., the same as `Pedersen`'s [24]. The worst case occurs when using linear search in the algorithms. Using binary search instead should bring complexity down to $\mathcal{O}(n \log n)$ in most realistic scenarios [24].

4.2 Experimental Validation

4.2.1 Experimental Setup

To compare `QBS` to `Pedersen`, we use the XWeB benchmark [15], which remodels the TPC-H [31] relational database as a star XML schema. XWeB initially generates documents scaling in size from 50,000 to 250,000 facts. The first and second rows of Table 1 range generated data in number of facts and kilobytes (mininum size is 13 MB and maximum size 67 MB).

Then, since XWeB's data warehouse is not modeled w.r.t. our data tree model (Section 3.3.1), we must translate it. Figure 11 depicts the XWeB data tree model, which contains *sale* facts, four dimensions (*part*, *customer*, *supplier* and *date*) and two measures (*f_quantity* and *f_totalamount*). The third row of Table 1 lists the sizes of the corresponding instances (minimum size is 27 MB and maximum size is 135 MB).

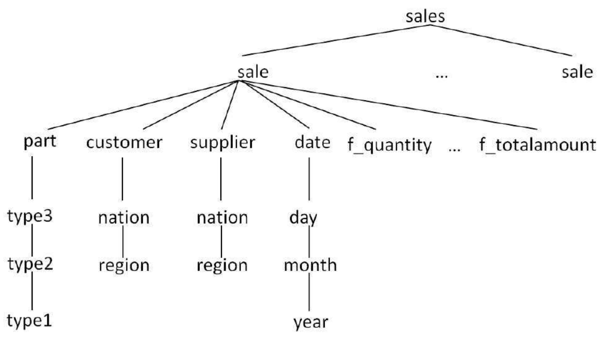


Figure 11: XWeB data tree model

Then again, XWeB's data warehouse does not include any complex hierarchy. Thus, we create variants of the

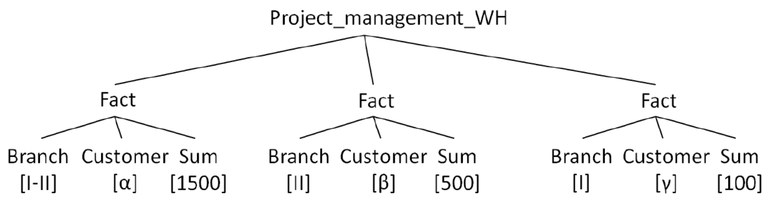


Figure 10: Sample roll up operation – Step #2

Table 1: Dataset size (KB)

No. Facts	50,000	100,000	150,000	200,000	250,000
XWeB	13,661	27,366	41,070	54,775	68,479
XWeB DT	27,700	55,390	82,800	110,577	138,015
Incomplete 5%	27,626	55,242	82,543	110,249	137,573
Non-strict 5%	28,669	57,328	85,671	114,422	142,786
Complex 5%	28,376	56,742	84,791	113,252	141,319
Incomplete 50%	25,020	50,030	74,769	99,842	124,601
Non-strict 50%	35,412	70,826	105,914	141,397	176,527
Complex 50%	32,522	65,031	97,263	129,839	162,088

dataset with different configurations of hierarchies: incomplete only, non-strict only and complex. Moreover, complexity is distributed by percentage of total number of dimensional nodes. For example, in Table 1, which features the sizes of these datasets (last six rows), "Complex 5%" on 50,000 facts means that among 200,000 dimensional nodes ($50,000 \times 4$ since each fact refers to four dimensions), 10,000 nodes are made complex. Such nodes are randomly distributed among every 20 (100/5) dimensional nodes. Moreover, the value of each generated node is randomly selected w.r.t. its dimensional applicable values, e.g., a *month* node must contain numerical values between 1 and 12. Note that although Table 1 shows data sizes only for the 5% and 50% configurations, we also exploit intermediate configurations, i.e., 10% and 20%. In Table 1, also note that data size expectingly decreases in incomplete configurations, since some subnodes are deleted, while data size increases in non-strict configurations, since subnodes are added to some dimensional nodes. Globally, data size increases in complex configurations since increases due to non-strictness are greater than decreases due to incompleteness.

Among XWeB's workload of queries, we focus on four queries with various number of dimensions (labeled 1D to 4D), and select the most detailed hierarchy levels for grouping because they form more complex groups. As shown in Table 2, we roll up to levels *day*, *type3*, *nation* and *nation* of dimensions *date*, *part*, *customer* and *supplier*, respectively. n represents the number of dimension involved in a given query. The *sum* aggregation function is used in our experiments to compute the total sale amount $sum(f\_totalamount)$. Any other aggregation function could be used, though.

Table 2: Group by n-dimensions queries

n	part	customer	supplier	date
1D				day
2D	type3			day
3D	type3	nation		day
4D	type3	nation	nation	day

Finally, our experiments run on a Toshiba laptop with an Intel(R) Core(TM) i7-2670QM CPU @ 2.20 GHz, 4 GB memory and 64-bit Windows 7 Home Premium, Service Pack 1. The `QBS`, `MakeCovering` and `MakeStrict` algorithms are implemented in Java JDK 1.7, using the SAX parser to read XML data. The only difference between our Java code and Algorithm 1 is that the output tree is built on the fly instead of applying a product on intermediary trees, to optimize performance.

4.3 Experimental Results

The following subsections present the results of our experimental comparison of `QBS` and `Pedersen`. To perform this comparison, we created metadata so that `MakeCovering` replaces incomplete values by "Other" like `QBS` does. For `Pedersen`, we also differentiate between query execution time and preprocessing overhead, while we cannot for `QBS` since it operates at query time and overhead is confused with query execution time.

4.3.1 Results on Simple Hierarchies

Figure 12 shows that `QBS`' time performance increases linearly with data size (i.e., the number of facts) and the number of dimensions in the query, except for query 3D on 50,000 facts, which incidentally bears a lower grouping complexity.

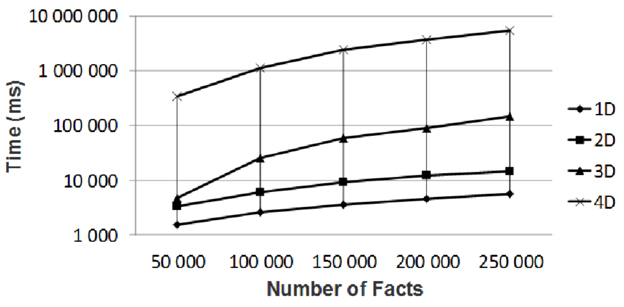


Figure 12: `QBS`' execution time according to the numbers of facts and of dimensions

Figure 13 shows that the time performance of both approaches increases linearly w.r.t. data size and the number of dimensions used in queries.

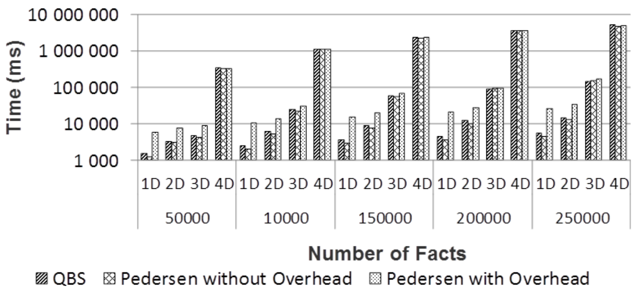


Figure 13: Comparison of `QBS` and `Pedersen` on simple hierarchies

On average, the execution time of `QBS` is 2 times lower than that of **`Pedersen with overhead`**, but it is 0.17 times higher than that of **`Pedersen without overhead`**.

However, both `QBS` and `Pedersen` consume a lot of time, especially when running the 4D query (about an hour). To find out why, we perform two more experiments, dissociating complex hierarchy processing time (i.e., summarizability processing time) from group matching time. This is possible because XWeB's data are originally summarizable. Figure 14 shows that enforcing summarizability in `QBS` does not affect time performance much, while group matching has a great impact that increases with the number of dimensions.

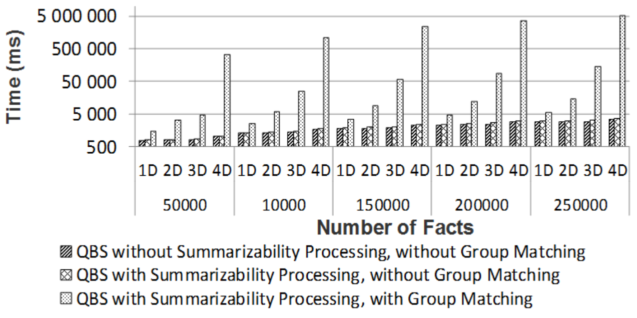


Figure 14: Comparison of summarizability processing time and group matching time in `QBS`

Figure 15 confirms that `Pedersen` also spends most of its time processing group matching, while overhead consumes little time. When processing group matching, we indeed need to check whether the group exists.

Thus, we must check every hierarchy level instance in the whole group, which contains several instances from all dimensions. Doing so is very time consuming comparing to traditional aggregation, which only checks for the existing group as a whole. However, no approach dealing with XML grouping, and *a fortiori* no XOLAP approach, can avoid this issue.

4.3.2 *Results on Complex Hierarchies*

Due to space limitations, we only present here our experiments on 5% and 50% incomplete, non-strict and complex hierarchies (the approximate minimum and maximum scale), but we did go through the whole range.

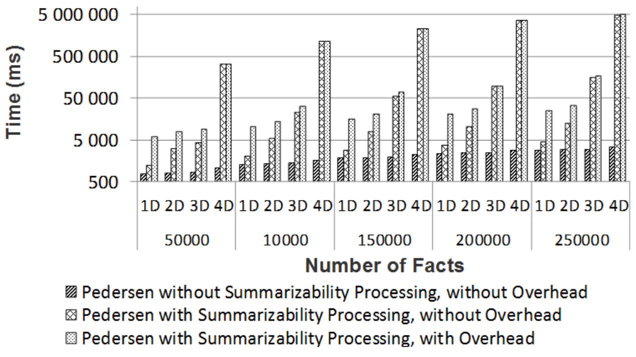


Figure 15: Comparison of summarizability processing time and group matching (overhead) time in `Pedersen`

4.3.2.1 *Incomplete Hierarchies.*

The results from Figures 16 and 17 reveal two cases. When the number of dimensions is small (up to query 2D), the execution time of `QBS` is 0.9 times lower than that of **`Pedersen with overhead`**, for both 5% and 50% hierarchies, on average.

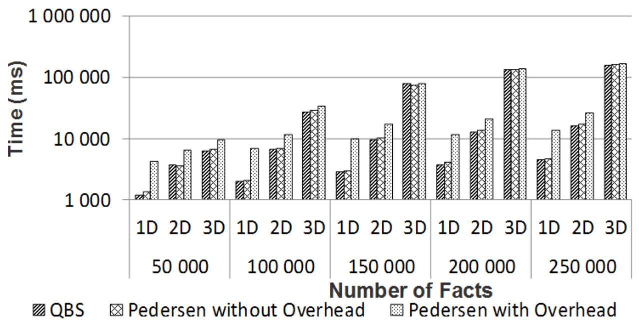


Figure 16: Comparison of `QBS` and `Pedersen` on 5% incomplete hierarchies

When overhead is not included in `Pedersen`, the execution time of `QBS` is 0.04 times lower (i.e., extremely close) on 5% hierarchies and 0.02 times lower (i.e., extremely close) on 50% hierarchies, on average. For a larger number of dimensions (query 3D), the execution time of `QBS` is the same as **`Pedersen without overhead`** on 5% hierarchies and 0.06 times lower (i.e., extremely close) than that of **`Pedersen without overhead`** on 50% hierarchies, on average. When overhead is included in `Pedersen`, `QBS'` execution time is on average 0.2 and 0.06 times lower (i.e., extremely close), on 5% and 50% hierarchies, respectively. Both approaches actually have different tradeoffs. `QBS` takes less time when reading incomplete data, but more time to solve incompleteness, while the reverse is true for `Pedersen` where data are normalized. Thus, when the number of dimensions increases, `QBS`' overhead when processing incomplete hierarchies at run-time is a handicap that evens global performances w.r.t. `Pedersen`. Still, we can notice that both approaches are affected by the poor performance of group matching, which explains why we did not include query 4D in these experiments.

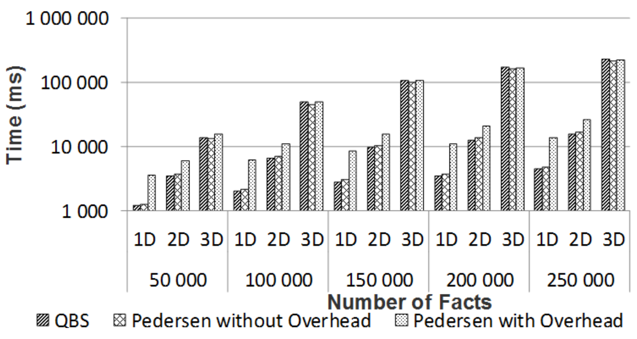


Figure 17: Comparison of QBS and Pedersen on 50% incomplete hierarchies

4.3.2.2 Non-Strict Hierarchies.

The results from Figures 18 and 19 show similar trends to those of Figures 16 and 17, because the tradeoffs in `QBS` and `Pedersen` are essentially the same for non-strictness management.

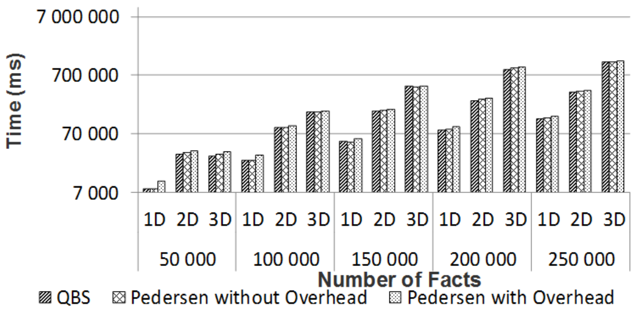


Figure 18: Comparison of QBS and Pedersen on 5% non-strict hierarchies

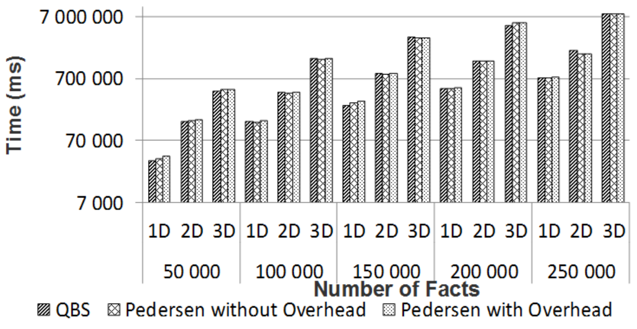


Figure 19: Comparison of QBS and Pedersen on 50% non-strict hierarchies

However, for `QBS`, non-strictness processing is 9 times higher than incompleteness processing, on average (Figure 20). Moreover, non-strictness processing is 37 times higher than incompleteness processing, on average (Figure 21).

Ultimately, the execution time of `QBS` is 0.1 times lower than that of `Pedersen with overhead` (5% hierarchies) and 0.03 times lower (i.e., extremely close) than that of `Pedersen with overhead`, on average (50% hierarchies). When overhead is not included in `Pedersen`, the execution time of `QBS` is on average 0.05 times lower (5% hierarchies) and 0.01 times lower (50% hierarchies) (i.e., extremely close).

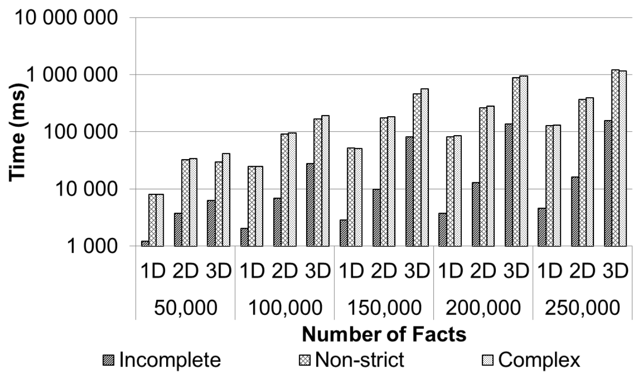


Figure 20: Evaluation of the three types of 5% hierarchies in QBS

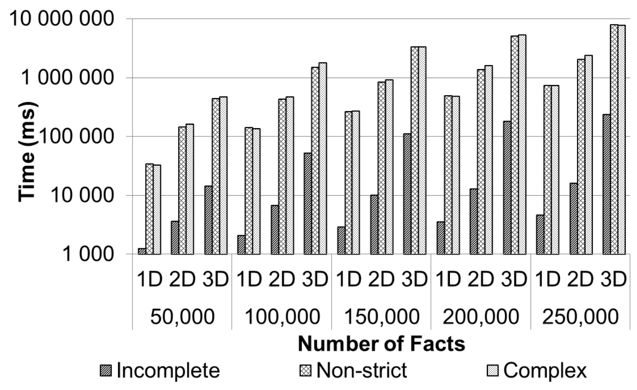


Figure 21: Evaluation of the three types of 50% hierarchies in QBS

4.3.2.3 Complex Hierarchies.

The results from Figures 22 and 23 bear similar results to the non-strict case, again because the cost of non-strictness processing is much higher than that of incompleteness processing (Figures 20 and 21).

Group matching is indeed mainly impacted by non-strict hierarchies. However, in some cases, such as in the 3D query on 250,000 facts in Figure 20, `QBS` performs better in the complex case than in the non-strict case, because non-strict processing incidentally produces fewer complex groups, thus simplifying group matching. For 5% hierarchies, `QBS'` execution time is 1.8 times lower than that of `Pedersen with overhead` and 0.01 times lower (i.e., extremely close) than that of `Pedersen without overhead`, on average. For 50% hierarchies, `QBS'` execution time is 0.09 times lower (i.e., extremely close) than that of `Pedersen with overhead` and 0.05 lower (i.e., extremely close) than that of `Pedersen without overhead`, on average.

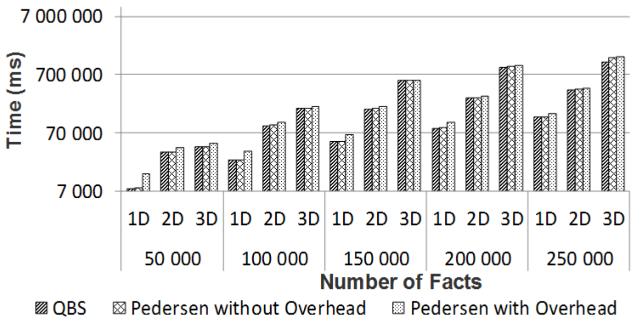


Figure 22: Comparison of QBS and Pedersen on 5% complex hierarchies

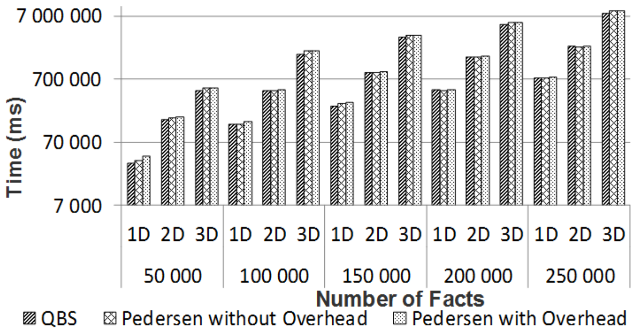


Figure 23: Comparison of QBS and Pedersen on 50% complex hierarchies

5. CONCLUSION AND PERSPECTIVES

In this paper, we propose the first truly operational query-based approach to solve summarizability issues in XML complex hierarchies. With respect to existing approaches, ours (1) modifies neither schema nor data, and thus has no space overhead and does not alter schema nor data semantics; (2) does not require any expertise beyond the user's, thus sparing the cost of expert intervention; (3) is dynamic w.r.t. schema and data evolution, thus favoring scalability.

We indeed experimentally demonstrate that the overhead induced by managing hierarchy complexity at run-time is totally acceptable. The performance, in terms of query response time, of our `QBS` algorithm is indeed comparable to that of Pedersen et al.'s reference algorithms. However, our comparison holds when the dataset is static. If schema or data updates were made, complex hierarchy processing would take place at regular intervals of time with `Pedersen` (instead of once in our experiments). By contrast, `QBS` would not have any further overhead, and should thus become more efficient.

Finally, our approach is implemented as a free Java prototype that is available online, along with our experimental datasets and the source code of the `QBS` and `Pedersen` algorithms[1].

The perspectives of this work are twofold. First, although XML is the best-suited format to represent complex hierarchy structures, our experiments show that summarizability management approaches are still too costly for realistic OLAP processing, which is supposed to run *online*, due to group matching cost. Thus, it is crucial to optimize the performance of our approach, e.g., by storing data in a non XML native fashion and/or using effective sorting, indexing and parallel processing techniques in group matching.

In a second step, we aim to define other XOLAP operators (cube, drill down, etc.) over complex hierarchies in order to complete an algebra, and implement them in our software prototype to provide a fully operational XOLAP framework.

[1]`http://eric.univ-lyon2.fr/~mhachicha/XOLAP.zip`

6. REFERENCES

[1] A. Abelló, J. Samos, and F. Saltor. YAM^2: a multidimensional conceptual model extending UML. *Information Systems*, 31(6):541–567, 2006.

[2] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon. XML Path Language (XPath) 2.0 (Second Edition). http://www.w3.org/TR/xpath20/, 2010.

[3] K. S. Beyer, D. D. Chamberlin, L. S. Colby, F. Özcan, H. Pirahesh, and Y. Xu. Extending XQuery for Analytics. In *24th International Conference on Management of Data (SIGMOD 05), Baltimore, USA*, pages 503–514, 2005.

[4] C. E. Dyreson, T. B. Pedersen, and C. S. Jensen. Incomplete Information in Multidimensional Databases. In M. Rafanelli, editor, *Multidimensional Databases: Problems and Solutions*, pages 282–309. Idea Group, 2003.

[5] M. Golfarelli, D. Maio, and S. Rizzi. The Dimensional Fact Model: A Conceptual Model for Data Warehouses. *International Journal of Cooperative Information Systems*, 7(2-3):215–247, 1998.

[6] M. Hachicha and J. Darmont. A Survey of XML Tree Patterns. *IEEE Transactions on Knowledge and Data Engineering*, 2012. In preprint.

[7] R. D. Hackathorn. *Web farming for the data warehouse*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, San Francisco, USA, 1999.

[8] J. Horner and I.-Y. Song. A Taxonomy of Inaccurate Summaries and Their Management in OLAP Systems. In *24th International Conference on Conceptual Modeling (ER 05), Klagenfurt, Austria*, volume 3716 of *LNCS*, pages 433–448. Springer, 2005.

[9] W. Hümmer, W. Lehner, A. Bauer, and L. Schlesinger. A Decathlon in Multidimensional Modeling: Open Issues and Some Solutions. In *4th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 02), Aix-en-Provence, France*, volume 2454 of *LNCS*, pages 275–285. Springer, 2002.

[10] C. A. Hurtado, C. Gutiérrez, and A. O. Mendelzon. Capturing Summarizability with Integrity Constraints in OLAP. *ACM Transactions on Database Systems*, 30(3):854–886, 2005.

[11] R. Kimball and M. Ross. *The Data Warehouse Toolkit*. John Wiley & Sons, second edition, 2002.

[12] J. Lechtenbörger and G. Vossen. Multidimensional Normal Forms for Data Warehouse Design. *Information Systems*, 28(5):415–434, 2003.

[13] H.-J. Lenz and A. Shoshani. Summarizability in OLAP and Statistical Data Bases. In *9th International Conference on Scientific and Statistical Database*

Management (SSDBM 97), Olympia, Washington, USA, pages 132–143. IEEE Computer Society, 1997.

[14] Z. Li, J. Sun, J. Zhao, and H. Yu. Transforming Non-covering Dimensions in OLAP. In *7th Asia-Pacific Conference (APWeb 05), Shanghai, China*, volume 3399 of *LNCS*, pages 381–393. Springer, 2005.

[15] H. Mahboubi and J. Darmont. XWeB: the XML Warehouse Benchmark. In *2nd TPC Technology Conference on Performance Evaluation & Benchmarking (TPCTC 10), Singapore*, volume 6417 of *LNCS*, pages 185–203. Springer, September 2011.

[16] E. Malinowski and E. Zimányi. Hierarchies in a multidimensional model: from conceptual modeling to logical representation. *Data & Knowledge Engineering*, 59(2):348–377, 2006.

[17] E. Malinowski and E. Zimányi. *Advanced Data Warehouse Design*. Springer, Berlin, Heidelberg, Germany, 2008.

[18] S. Mansmann and M. H. Scholl. Extending Visual OLAP for Handling Irregular Dimensional Hierarchies. In *8th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 06), Krakow, Poland*, volume 4081 of *LNCS*, pages 95–105. Springer, 2006.

[19] S. Mansmann and M. H. Scholl. Empowering the OLAP Technology to Support Complex Dimension Hierarchies. *International Journal of Data Warehousing and Mining*, 3(4):31–50, 2007.

[20] J.-N. Mazón, J. Lechtenbörger, and J. Trujillo. Solving Summarizability Problems in Fact-Dimension Relationships for Multidimensional Models. In *ACM 11th International Workshop on Data Warehousing and OLAP (DOLAP 08), Napa Valley, USA*, pages 57–64, 2008.

[21] J.-N. Mazón, J. Lechtenbörger, and J. Trujillo. A survey on summarizability issues in multidimensional modeling. *Data & Knowledge Engineering*, 68(12):1452–1469, 2009.

[22] D. Pedersen, J. Pedersen, and T. B. Pedersen. Integrating XML Data in the TARGIT OLAP System. In *20th International Conference on Data Engineering (ICDE 04), Boston, USA*, pages 778–781. IEEE Computer Society, 2004.

[23] D. Pedersen, K. Riis, and T. B. Pedersen. A Powerful and SQL-Compatible Data Model and Query Language for OLAP. In *13th Australasian Database Conference (ADC 02), Melbourne, Australia*, volume 5 of *CRPIT*. Australian Computer Society, 2002.

[24] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Extending Practical Pre-Aggregation in On-Line Analytical Processing. In *25th International Conference on Very Large Data Bases (VLDB 99), Edinburgh, Scotland, UK*, pages 663–674. Morgan Kaufmann, 1999.

[25] E. Pourabbas and M. Rafanelli. Hierarchies and Relative Operators in the OLAP Environment. *SIGMOD Record*, 29(1):32–37, 2000.

[26] M. Rafanelli and A. Shoshani. STORM: A Statistical Object Representation Model. In *5th International Conference on Statistical and Scientific Database Management (SSDBM 90), Charlotte, NC, USA*, volume 420 of *LNCS*. Springer, 1990.

[27] S. Rizzi. Conceptual Modeling Solutions for the Data Warehouse. In R. Wrembel and E. Christian Koncilia, editors, *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, pages 1–26. IRM Press, Hershey, USA, 2007.

[28] S. Rizzi, A. Abelló, J. Lechtenbörger, and J. Trujillo. Research in data warehouse modeling and design: dead or alive? In *ACM 9th International Workshop on Data Warehousing and OLAP (DOLAP 06), Arlington, Virginia, USA*, pages 3–10. ACM, 2006.

[29] I.-Y. Song, W. Rowen, C. Medsker, and E. F. Ewen. An Analysis of Many-to-Many Relationships Between Fact and Dimension Tables in Dimensional Modeling. In *3rd International Workshop on Design and Management of Data Warehouses (DMDW 01), Interlaken, Switzerland*, volume 39 of *CEUR Workshop Proceedings*, page 6. CEUR-WS.org, 2001.

[30] R. Torlone. Conceptual Multidimensional Models. In E. Maurizio Rafanelli, editor, *Multidimensional Databases: Problems and Solutions*, pages 69–90. IDEA Group Publishing, Hershey, USA, 2003.

[31] TPC. *TPC Benchmark H Standard Specification revision 2.3.0*. Transaction Processing Performance Council, August 2005.

Hybrid HBase: Leveraging Flash SSDs to Improve Cost per Throughput of HBase

Anurag Awasthi
Dept. of Computer Science and Engineering,
Indian Institute of Technology, Kanpur, India
anuraga@cse.iitk.ac.in

Avani Nandini
Dept. of Computer Science and Engineering,
Indian Institute of Technology, Kanpur, India
nadini@cse.iitk.ac.in

Arnab Bhattacharya
Dept. of Computer Science and Engineering,
Indian Institute of Technology, Kanpur, India
arnabb@iitk.ac.in

Priya Sehgal
NetApp Corporation, India
priya.sehgal@netapp.com

ABSTRACT

Column-oriented data stores, such as BigTable and HBase, have successfully paved the way for managing large key-value datasets with random accesses. At the same time, the declining cost of flash SSDs have enabled their use in several applications including large databases. In this paper, we explore the feasibility of introducing flash SSDs for HBase. Since storing the entire user data is infeasible due to impractically large costs, we perform a qualitative and supporting quantitative assessment of the implications of storing the system components of HBase in flash SSDs. Our proposed HYBRID HBASE system performs 1.5-2 times better than a complete disk-based system on the YCSB benchmark workloads. This increase in performance comes at a relatively low cost overhead. Consequently, Hybrid HBase exhibits the best performance in terms of cost per throughput when compared to either a complete HDD-based or a complete flash SSD-based system.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query Processing and Optimization*

Keywords

HBase, Flash SSD, Big Data, Cost per Throughput

1. INTRODUCTION

Column-oriented databases have been proven to be well-suited for large database applications including data warehouses and sparse data [1]. Recently, there is a substantial interest in distributed data stores for large chunks of data, specially in the NoSQL domain, such as Google's BigTable [3], Amazon's Dynamo [6], Apache HBase [8] and Apache Cassandra [13]. These are being widely used by several companies and industrial users to store "big data" of the order of terabytes and petabytes on a daily basis. These systems are of *key-value store* type that utilize the column-oriented architecture.

Out of these, we choose to work with HBase for multiple reasons: (i) it is an open-source software and, therefore, easy to modify, (ii) it has been successfully deployed in many enterprises, (iii) it is capable of efficiently hosting very large data with tables having billions of rows and millions of columns including sparse data, and (iv) it has become increasingly popular in recent years and has a significantly large community following.

Traditionally, the column-oriented database systems have been designed considering disk (HDD) as the underlying storage media. This means that generally only random seeks have been attempted to be minimized. The lower latency involved in random reads in comparison to HDDs has drawn attention, and coupled with the reducing cost of flash drives and increasing capacity per drive, several successful attempts have been made for improving query performance by introducing flash SSDs (some well known examples are [7, 16, 20]). The use of flash SSDs as a substitute as well as a complementary storage media for hard disks is also increasing due to their lower power consumption, lower cooling cost, lesser noise and smaller sizes.

However, flash has certain disadvantages as well. While exhibiting good performance for random reads, it suffers in case of random writes. Flash SSDs do not allow in-place updates and requires subsequent garbage collection which results in write amplification and erasures overhead, thereby impacting random write performance. Frequent erase operations also shorten the lifetime of SSDs as flash devices can typically sustain only 10,000 to 100,000 erase cycles. This adversely affects the overall reliability of the SSD drive. Further, the cost per unit capacity of flash SSDs is approximately 10 times that of HDDs.

With such high costs, low density, and low reliability compared to hard drives, it is impractical to completely replace HDDs with flash SSDs in large deployments like databases (100s of terabytes to petabytes of capacity requirement). Instead, practitioners have adopted hybrid solutions consisting of a mix of SSD and HDD with different media serving different purposes – SSDs offering high throughput (measured in terms of I/O operations per second) while HDDs offering high storage capacity. Such hybrid solutions provide good performance at better costs compared to pure HDD or pure SSD systems [12].

In this work, we leverage this hybrid approach to come up with a better cost per throughput solution for HBase columnar database systems. As HBase has a lot of metadata or system components, we try to figure out the relevant items that should be placed in flash as opposed to HDD to yield an attractive cost per throughput. We call this modified HBase as HYBRID HBASE.

Hybrid SSD and HDD solutions come in two forms with SSD used as either as (i) a read-write cache for HDD [10, 24], or as (ii) a permanent store at the same level as HDD [4, 12, 15]. While the first case of using SSD as an intermediate tier between DRAM and HDD seems very simple to use and deploy, it leads to caching problems like redundancy, cache coherency (in case of shared HDD infrastructure), etc. Further, as flash is limited in its erase and program cycles, using it as a cache hurts its lifetime much more, thereby increasing the overall cost per unit capacity of the hybrid solution. Hence, we propose to use SSD as a permanent store at the *same* memory hierarchy as the HDD for our Hybrid HBase.

Any column-oriented database system has two main components residing on storage media: (i) user data components that store the actual data, and (ii) system components needed for user data management that include catalog tables, logs, temporary storage or other components storing information about current state of system, etc. While flash can be used to host both the components, for industrial strength data stores where data sizes are in the order of terabytes and petabytes, it may be infeasible to host the user data components due to impractically large costs. Further, the gain in throughput will depend heavily on access patterns, etc.

Thus, we focus only on hosting the system components of a large key-store data store on flash. In addition to being much smaller in size, system components do not change significantly with different database sizes and access patterns. For example, since write-ahead log is designed to have sequential I/O, it will be accessed sequentially irrespective of whether the update operation is a random update or a sequential update. Also, the size of the write-ahead log remains of the order of gigabytes even under heavy load. Additionally, system components must reside on a *persistent* media so that they can be retrieved after a system crash. This rules out the possibility of hosting them on main memory.

In this paper, we estimate which system components to host in the flash to improve the cost per throughput of the system. We identify the system components for a HBase system and analyze the effects of migrating them to flash both analytically as well as empirically (by performing a thorough benchmarking using the YCSB workloads). Since flash is used to host only a small amount of data, the increase in cost is low, although the improvement in throughput is quite high. Overall, this improves the cost per throughput of the system considerably as compared to a complete HDD-based setup or a complete flash SSD-based setup.

The focus of our proposed system is three-fold: (i) better cost per throughput, (ii) performance independent of access pattern, hit ratio, and size of data, (iii) easy to setup, i.e., easy deployment and migration from standard HBase system. Further, the approach presented is generic and can be applied to other column store architectures after similar analyses.

Specifically, the contributions of this paper are:

- We analyze the significance of the storage media in the performance of HBase. We assess disk and flash as storage media, and compare changes in performance with changes in system cost.

- We propose Hybrid HBase, which uses a combination of HDD (for data components) and flash SSD (for system components), and analyze its performance gain and system cost.

Parameter	Disk	Flash
Model	Western Digital wd10EARS	Kingston SV100S2
Capacity (GB)	1024	128
Cost/GB	$0.15	$2.00
Random Seeks (/s)	151	1460
Reads (MB/s)	161	307.5
Sequential writes (MB/s)	128	182.5
Random re-writes (MB/s)	63.2	81.63

Table 1: Different parameters of the two storage media.

The generic analysis can be extended to other column stores for improving the cost per unit throughput.

The rest of the paper is organized as follows. Section 2 presents the required background information needed to understand the idiosyncrasies of flash SSD as a storage media and HBase as a data store. It also briefly describes the related work. Section 3 discusses the feasibility of using flash SSD for hosting the system components of HBase and proposes the Hybrid HBase system. Section 4 describes the experimental setup along with performance comparison of the Hybrid HBase system against a complete HDD-based setup and a complete flash SSD-based setup. Finally, Section 5 concludes and outlines some possible future work.

2. BACKGROUND AND RELATED WORK

2.1 Flash as Storage Media

Hard disk drives (HDDs) are electromagnetic devices that have moving heads that read/write data using rotation of spindles. This enforces a mechanical bottleneck for I/O operations. In contrast, flash solid state devices (SSDs) does not contain any moving parts and provide instant reads. Consequently, flash provides good latencies for random reads in comparison to disks (up to 100 times for enterprise SSDs). Re-writes are slower in comparison to reads due to the *erase-before-write* mechanism where re-writing requires erasing a complete block after persisting all its data to a new location, leading to *write amplification*. This, therefore, results in asymmetric read and write performance. Further, each block can be erased only a finite number of times before it turns into a bad block (non-usable). Due to this erase-before-write mechanism, efficient wear leveling mechanism and garbage collection need to be supported on flash, else some blocks become unusable much earlier than others. Flash can, however, offer good performance for sequential writes. Also, it requires less power consumption. Performance comparison of HDDs versus flash SSDs have been done in [21, 22]. As illustrated in Table 1, the comparison of actual runtime parameters between disk and flash for the models used in our experiments shows the same trends.

2.2 HBase

Apache HBase[1] is an open-source implementation of Google's BigTable [3]. It is a distributed column-based key-value storage system that leverages existing open-source systems such as Zookeeper[2] and Hadoop's Distributed File System (HDFS)[3].

HBase *cluster* has one *master server*, multiple *region servers* and the client API. Zookeeper assists the master server in coordinating with the region servers.

Tables are generally sparse and contains multiple rows containing several columns, grouped together into *column-families*. All

[1]http://hbase.apache.org/
[2]http://zookeeper.apache.org/
[3]http://hadoop.apache.org/

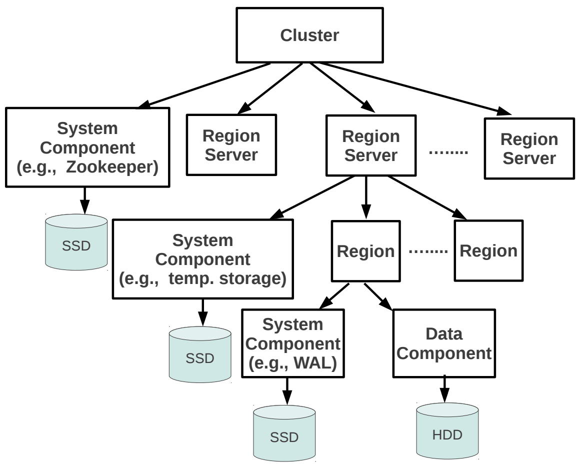

Cluster
System Component (e.g., Zookeeper)
Region Server
Region Server
....
Region Server
SSD
System Component (e.g., temp. storage)
Region
....
Region
SSD
System Component (e.g., WAL)
Data Component
SSD
HDD

Figure 1: Hybrid HBase setup.

columns of a column family are stored together in sorted *key-value* (ordered by key) format in *store files*. Each store file stores key-value pair corresponding to only one column-family.

Each region server can host several *regions*. A region is a horizontal division of a table and contains store files corresponding to all column-families of that division. A region splits horizontally (based on row key) into two daughter regions if its size grows above a threshold. Therefore, a table is comprised of multiple regions distributed over different region servers.

Each region server also has a *write-ahead log (WAL)* file shared by all its regions. When a write request from a client reaches a region server, data is first written persistently to the WAL and then to the in-memory *memstore*. The write-ahead log is used to retrieve the data after a server crash. After each flush, the write-ahead log can be discarded up to the last persisted modification.

The memstore stores data in a sorted manner, and its size can grow to the order of gigabytes. Once the size of memstore crosses a threshold, it is flushed to disk as a *store file* in a rolling fashion, i.e., HBase stores data residing on disk in a fashion similar to *log-structured merge (LSM) trees* [19], more specifically in "log-structured sort-and-merge-map" form as explained in [8]. According to [8], background compaction of store files in HBase corresponds to the merges in LSM trees and happens on a store file level instead of the partial tree updates. Therefore, HBase uses a *write-behind* mechanism and internally converts multiple random writes to a sequential write for large chunks of data.

To read a key-value pair, first the region server hosting the corresponding region is identified using *catalog tables*. At the region server, first the memstore is searched to see if the required value is present there. If not, then the next level of LSM tree stored persistently needs to be examined. This process continues until either all the levels of LSM trees have been examined or the key is found.

Write involves inserting the updated or new key-value pair in memstore and writing it sequentially to a WAL. Compaction, memstore flush and other such operations happen in background.

Therefore, in HBase, read latencies are higher than write latencies as a read requires first searching the memstore, followed by searching on-disk LSM-trees from the top most level to the bottom level in a merging fashion.

On the administrative side, all the information about regions and region servers are hosted in two *catalog tables* called *.META.* and *-ROOT-*. Zookeeper, which stores information about the region server, hosts the *-ROOT-* table. The *-ROOT-* table gives the address of the server hosting the *.META.* table which, in turn, contains the list of region servers and regions that they are hosting.

A new client first contacts Zookeeper to retrieve the server name hosting the *-ROOT-* table. Afterwards, these catalog tables are queried by the clients to reach the region server directly.

Only when catalog tables are changed due to system crash, region splitting, region merging or load balancing, does the client need to re-establish the connection. It is important to note that for most workloads such events are not too frequent. Thus, the catalog tables are mostly read-intensive entities. Further, although Zookeeper is extremely I/O intensive, it needs only a small amount of persistent data.

2.3 Related Work

Flash SSDs have been successfully used as storage media in many embedded systems and are ubiquitous in devices such as cell phones and digital cameras. Hybrid database systems using both types of storage media (i.e., HDDs and flash SSDs) have also been proposed [12, 24]. In [12], capacity planning technique was proposed to minimize the cost of a hybrid storage media. It uses flash SSDs as a complementary device for HDDs rather than a replacement. Further, in [24], a novel multi-tier compaction algorithm was designed. An efficient tablet server storage architecture that extends the Cassandra SAMT structure was proposed. It was shown to be capable of exploiting any layered mix of storage devices.

In [2], a flash-friendly data layout was proposed that used flash to boost the performance for DRAM-resident, flash-resident and HDD-resident data stores. Flash has also been used as part of a memory hierarchy (in between RAM and HDD) for query processing. In [9, 25], a general pipelined join algorithm was introduced that used a column-based page layout for flash. In [10] flash was used as a streaming buffer between DRAM and disk to save energy.

In order for applications to work transparently to the idiosyncrasies of the flash SSD media, various flash specific file systems have been developed. YAFFS [18] and JFFS[4] are among the most popular ones and are part of the log-structured file system (LFS) [23] class. LFS file systems has an advantage on flash as they log the changes made to the data instead of overwriting it, thereby trading the costly erase operations with increased number of read operations. LGeDBMS [11] used the design principle of LFS further and introduced log structure to flash-based DBMS.

In OLTP systems, significance of flash becomes evident due to the work of [15]. An order of magnitude improvement was observed in transaction throughput by shifting the transactional logs and roll back segments to flash SSD. An improvement by an order of two was also observed in sort-merge algorithms by using flash SSD for temporary tables storage. Further, in [14], it has been shown that flash SSDs can help reduce the gap between the increasing processor bandwidth and I/O bandwidth.

The work presented here is different from others due to multiple reasons. Firstly, there have been attempts to introduce flash in the memory hierarchy between RAM and disk as in [24], but to the best of our knowledge there is no work done for benchmarking the performance of column stores such as HBase with respect to flash SSD as storage media. Secondly, we focus on and explore the feasibility of using flash SSDs at the *same* memory hierarchy as disk for hosting system components. Thirdly, our approach can be generalized for any distributed key-value column-oriented storage system, in particular the NoSQL domain.

[4]http://sourceware.org/jffs2/jffs2-html/

3. THE HYBRID HBASE SYSTEM

In this section, we describe our Hybrid HBase system. The analyses of flash SSDs and HBase done in Section 2.1 and Section 2.2 respectively suggest that it is beneficial to leverage flash SSDs for setting up a HBase system. However, when storage requirements are high, it is not feasible to replace the entire storage capacity of HDDs by flash SSDs. Hence, we focus *only* on the system components of HBase.

3.1 System Components

The major system components of HBase are:

- Zookeeper data
- Catalog tables (*-ROOT-* and *.META.*)
- Write-ahead logs (WAL)
- Temporary storage for compaction and other such operations

In the following sections, for each of the above mentioned system components, we discuss analytically whether hosting it on flash SSD can give any performance boost. Section 4.2 analyzes the empirical effects of putting them on a flash SSD as opposed to a HDD.

3.1.1 Zookeeper

The Zookeeper data component stores information about the master server as well as the region server hosting the *-ROOT-* table, in addition to a list of alive region servers. The client contacts the Zookeeper to retrieve the server hosting the *-ROOT-* table while the master contacts it to know about the available region servers. The region servers report to Zookeeper periodically to confirm their availability. This is similar to a heartbeat keep-alive mechanism and a region server would be declared unavailable if it fails to report. This, thus, makes the Zookeeper very I/O intensive.

The storage requirements for Zookeeper is essentially proportional to the number of systems in the HBase cluster. For most cases, it is very low and is in the order of kilobytes only per system. Hence, it should be beneficial to host it in a flash SSD. However, it cannot be hosted on main memory due to persistency requirements.

3.1.2 Catalog Tables

The catalog tables (*-ROOT-* and *.META.*) are mostly read intensive and are not updated as frequently as the data tables. While the *-ROOT-* table has almost a fixed size, the size of the *.META.* table grows with the total number of regions in the cluster. Nevertheless, their sizes are much less (almost insignificant) in comparison to the data. Thus, these tables are also good candidates for being hosting on flash SSDs. Again, although the sizes of these tables can fit into main memory, they cannot be hosted there as persistency needs to be maintained across system crashes.

3.1.3 Write-ahead-log (WAL)

The write-ahead-log (WAL) is used to simulate sequential writes. Any write is first done on the WAL and it is later committed to the disk in a rolling fashion. The WAL itself is written in a sequential manner as well.

The size of the WAL, unlike the other system components, is not small. The size grows proportionately with the following three parameters: (i) the time after which the WAL is committed to disk, (ii) the rate at which writes happen, and (iii) the size of each key-value pair. Thus, depending on the workload, the size can become as large as gigabytes. This, therefore, rules out the possibility of using main memory. Also, if the WAL resides on a flash SSD, system recovery would be faster after a system crash as data written in WAL could be read faster from SSDs. Hence, it would be productive to host it on flash SSDs.

3.1.4 Temporary Storage

Temporary storage space is used when a region is split or merged. The rows are generally written sequentially in the temporary storage and then later read in a sequential manner again. The size is not expected to be large unless there are many region splits and merges. Combined with the sequential nature of access, introducing flash for temporary storage should, thus, improve the performance.

The above analyses thus suggest that shifting all the four system components of HBase to flash SSDs can yield a better performance at a marginal cost overhead. (Section 4.2 shows the gain in throughput for each system component individually.) This forms the basis of our proposed HYBRID HBASE system. The setup is shown schematically in Figure 1. We next estimate the additional cost of such a hybrid system.

3.2 Additional Cost of Hybrid HBase

The overhead of catalog tables is directly related to the size of the database. If the maximum number of keys per region (as configured by the HBase administrator) is R, then the number of entries in *.META.* is $m = N/R$, where N is the total number of records in the database in a stable *major compacted* state. The *-ROOT-* in turn contains only m/R entries. Thus, we need extra space in the order of $1/R + 1/R^2$ times the user data space. For typical values of R, e.g., when $R = 1000$, this translates to an overhead of only $\approx 0.1\%$.

The space overheads for the Zookeeper and the temporary directory are proportional to the number of systems in cluster and are insignificant in comparison to the total size of the database.

The write-ahead-log (WAL), however, can grow to a significant size, and a flash SSD needs to be installed on each region server. To get an upper estimate of the size of WAL, we observe that in the worst case all the memstores will remain uncommitted and the WAL will keep on growing. Usually there is an upper limit on the size of the memstores and is always less than the heap size allocated to HBase. However, in the extreme case, the entire heap may be used for this purpose (although not recommended), thereby starving other processes. This allows us to estimate the upper limit by the size of the heap allocated for HBase. For our experiments, we used 4 GB of heap and a maximum of 2 GB of memstores before flushing is forced. Even in higher end server machines having 32 GB RAM, if 16 GB is devoted for WAL (which is a high estimate)[5], we only need a flash SSD partition of size 16 GB on each region server. The user data hosted on these machines can be very high (say up to 2-4 TB) without increasing the risk of over-running WAL. Thus, this constitutes the largest system cost requirement. Assuming a 1 TB database and a 8 GB WAL space, the cost overhead is $8/1024 \approx 0.8\%$.

Adding all the system components together, the space overhead grows to at most 1% of the total database size. At an estimate of flash SSDs being 10 times more expensive than HDDs, the extra cost overhead of our proposed Hybrid HBase system for installing flash SSD drives is 10%. Thus, if the gain in throughput becomes more than 10%, then the cost per unit throughput of the hybrid system would be better.

Section 4 extensively discusses the gain in throughput by using flash SSDs. However, before we present the experimental results on how the hybrid system fares vis-à-vis a completely HDD based system or a completely flash SSD based system, we describe our

[5]It is better to flush WAL when the size is small as then the system rollback and recovery are faster after a system crash.

Workload Name	Operations	Access Pattern
A−Update heavy	Read: 50% Update: 50%	Zipfian
B−Read heavy	Read: 95% Update: 5%	Zipfian
C−Read only	Read: 100%	Zipfian
D−Read latest	Read: 95% Insert: 5%	Latest
E−Short ranges	Scan: 95% Insert: 5%	Zipfian/ Uniform
F−Read-modify-write	Read: 50% Read-Modify-Write: 50%	Zipfian

Table 2: YCSB workloads, as published in [5].

model of how the systems are compared according to the cost and the cost per unit throughput measures.

3.3 Metrics for Comparing Systems

We compare the cost of storage media only as this is the sole component varying across different system setups. In addition to a fixed installation cost, there is a maintenance cost associated with each storage media that includes power usage, cooling cost and other such recurring costs. However, since it is harder to estimate them and manage them, in this paper, we only consider the installation cost, information about which is readily available.

To calculate the system cost for a storage media over a given workload, we first estimate the maximum amount of data stored in the device while the workload is running. We also set the device utilization ratio to 80% for HDDs and 50% for flash SSDs as suggested in [12]. The device utilization ratio is important as when the data size grows above it, the performance of the media decreases due to various factors including garbage collection.

Assume that a system setup S uses n storage media. The maximum capacity and the utilization ratio for each of them are $\{D_1, D_2, \ldots, D_n\}$ and $\{R_1, R_2, \ldots, R_n\}$ respectively. Hence, the amount of data that can be stored in a device i is only D_i/R_i. If the price for unit capacity of each storage media is $\{P_1, P_2, \ldots, P_n\}$, the system cost C for the entire setup S is

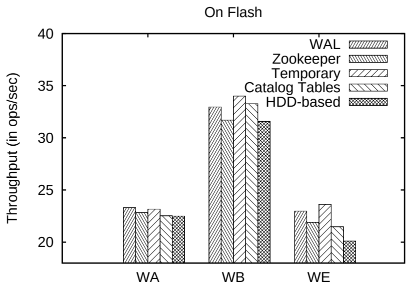
$$C = \sum_{i=1}^{n} (P_i.D_i/R_i)$$

However, due to significant differences in latencies and cost of the three systems (the hybrid one and the two using only one type of storage media), we use the *cost per unit throughput* metric for a fair comparison. If a system having a cost of C achieves a throughput of T IOPS (I/O operations per sec), the cost per unit throughput is C/T.

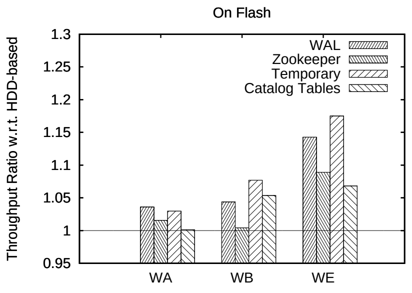
4. EXPERIMENTAL EVALUATION

In this section, we present the experimental analysis and benchmarking of our proposed hybrid system vis-à-vis a complete flash-based system and a complete disk-based system. We conduct the experiments on a standalone instance of HBase (similar to [24]) to completely eliminate the network related latencies. This enables us to better understand the performance and design implications of Hybrid HBase. Since the idea is to analyze performance improvement with respect to storage media, we can expect gain in performance in similar proportions for a distributed environment.

The results are reported for experiments on a system running on an Intel i5-2320 LGA1155 processor (4 cores and 4 threads at 3 GHz) with a total of 8 GB of RAM (4 GB as heap), Western Digital 1 TB HDD, Kingston SV100S2 128 GB Flash SSD, with 64-bit Ubuntu-Server 11.10 as the operating system and ext4 as the underlying file system. We used HBase version 0.90.5 from the Apache repository as the base system. For all analysis and performance evaluations, we used Yahoo! Cloud Serving Benchmarking (YCSB) [5] version 0.1.4. Table 2 shows the six standard workloads (A to F) as identified in [5].



(a) Raw throughputs



(b) Throughput as a ratio with HDD

Figure 2: Throughputs when single system components are hosted on flash SSD.

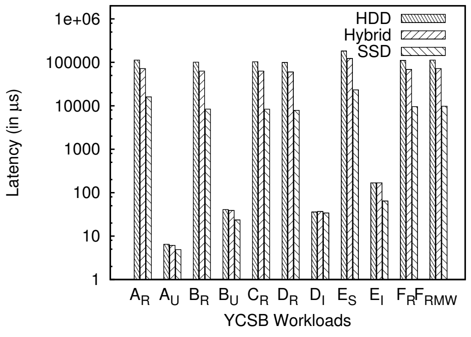
The workloads are composed of Q number of queries (or operations) on R records, and the key generation pattern is decided by three models, namely, latest, uniform and Zipfian. For a workload following a uniform distribution, all records in the database are equally likely to be chosen for the next query. For a Zipfian distribution, some randomly selected keys are hot (more frequently accessed) while most records are rarely accessed for queries. Latest distribution, as the name implicates, reads or writes the most recently accessed key-value pairs with a higher probability.

For our analysis, we used $Q = 10^6$ queries on a database with $R = 6 \times 10^7$ records. Each record is of size 1 KB and the total number of regions in a compact state was found to be 72 (with a maximum region size of $\approx$ 1 GB). We next discuss a few important parameters of the system and the HBase configuration.

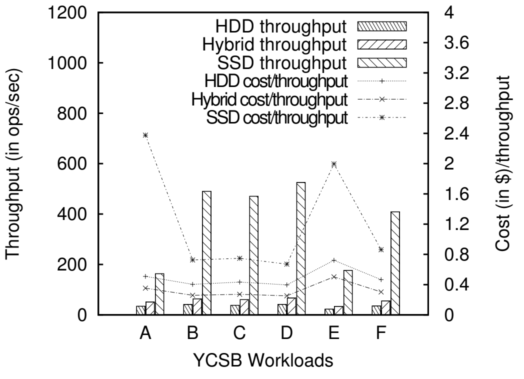
4.1 System Tuning

The benchmarking of any given system involves several variables which must be taken care of appropriately to get the true effect of the desired variable, which in our case, is the storage media. By considering a standalone system, we have removed all external network related issues. We run HBase on a dedicated partition which is different from the operating system's (O/S) partition. The O/S runs on an ext4 HDD partition. Out of 8 GB RAM available, 4 GB had been allocated as heap for HBase and 4 GB had been used by O/S. We also set the swappiness[6] parameter to zero to enable using the entire available RAM. For the ext4 file system, we

[6]Swappiness is the tendency to use swap area in place of RAM in order to reserve some RAM for future processes.



(a) Average Latency[8]



(b) Throughput

Figure 3: Performance over different YCSB workloads.

deactivated the maintenance of file access times done by kernel to further reduce the administrative overheads not needed by HBase.

On flash SSD, we additionally enable TRIM[7] support to reset all flash SSD wear-leveling tables prior to evaluation and maintain a 50% utilization ratio. This minimizes the internal flash SSD firmware interference due to physical media degradation and caching and enhances the flash performance. An unused flash performs very well for the initial read and writes, before reaching a stable lower performance. Hence, we completely fill and empty the flash several times to eliminate this effect. Further, before starting the experiments, we fill SSD completely with some random data so that each query has the same state of flash for garbage collection.

For HBase, automatic major compaction was disabled. We perform major compaction manually and also empty the cache before each experiment to provide the same *data locality*, i.e., the same initial state for both cache and the data layout on disk. The MSLAB [17] feature has been enabled to facilitate garbage collection as well as to avoid lengthy pauses and memory fragmentation due to write heavy workloads. We set the maximum regions per server to 200 and extended the session timeout limit (after which a server is declared dead) to avoid possible server crashes due to delay in responses when the system is subjected to an overload.

[7]The TRIM command specifies which blocks of data in an SSD are no longer used and can be erased.

[8]$X_R \equiv$ Read operation of workload X; $X_U \equiv$ Update operation of workload X; $X_S \equiv$ Scan operation of workload X; $X_I \equiv$ Insert operation of workload X; $X_{RMW} \equiv$ Read-modify-write operation of workload X.

4.2 Single Component Migration

Before we benchmark the proposed Hybrid HBase system, we first assess the effect of migrating one system component at a time. These experiments, thus, measure the effects of hosting each system component *individually* on a flash SSD while the rest three *remain* on the HDD.

We ran half a million (5×10^5) queries on a database having 60 million (6×10^7) records over the workloads WA, WB and WE, i.e., update-heavy, read-heavy and short-ranges. The characteristics of the other workloads are similar to these (WC and WD are both read-heavy and are similar to WB while WF has 50% read and 50% write, similar to what WA also has).

Figure 2 shows the throughputs of the setups (both raw and as a ratio with a completely HDD-based system). The gains in throughput are more pronounced for WAL and temporary storage. Hence, hosting these components on flash SSD is likely to improve the cost per throughput ratio. However, since the space (and therefore, cost) overheads of the catalog tables and Zookeeper are almost insignificant, it is beneficial to host them on flash SSDs as well. These conclusions, therefore, agree with the analyses done in Section 3.1.

4.3 Performance over the YCSB Workloads

Figure 3 depicts the performance of the Hybrid HBase setup vis-à-vis the completely HDD-based system and the completely flash SSD-based system for the different operations on the six YCSB workloads. (As mentioned earlier, for all subsequent experiments, the database consists of 6×10^7 keys and results reported are averages over 3 runs, each having 10^6 queries. Moreover, all the four system components are hosted on a flash SSD.)

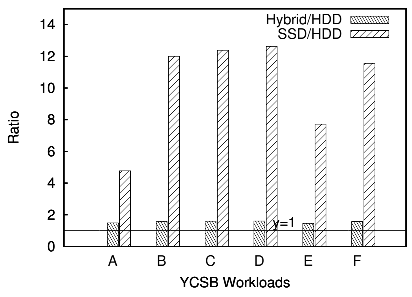
Read latencies of SSD-based setup are significantly lower (approximately 13 times) than both Hybrid and HDD-based setups. These read operations are random reads which are significantly faster for a flash SSD and, hence, the lower latencies. Since the catalog tables (*-ROOT-* and *.META.*) and also the Zookeeper data is stored on SSD in the hybrid setup, read latencies are lower than HDD (approximately 1.6 times). The user data remains on the disk, and therefore, latencies are not as low as SSD.

Average latency for update operation is the lowest for SSD followed by Hybrid and is the highest for HDD. The update operation is similar to a random write, and thus, involves writing to the write-ahead-log (WAL) persistently and storing the updates in memstores to be flushed later. Since WAL is on flash SSD in a Hybrid setup, average update latencies for Hybrid and SSD should be similar. However, due to other background processes (e.g., major compaction and JVM garbage collection) that run faster in SSD, the update latencies for SSD setup are lower.

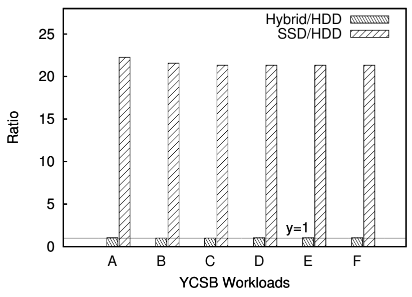
SSD outperforms Hybrid and HDD setup in scans (sequential reads) moderately as the difference between sequential reads for HDD and flash SSD is not as high as random reads (see Table 1). Average insert latency for HDD, Hybrid and SSDs are also almost similar. Insert operation differs from update operation as during inserts, the size of a region grows and may lead to a region split. A region split also requires updating the *.META.* table. Thus, average insert latency is higher than average update latency over different workloads.

Overall, therefore, as expected, the throughputs of a completely SSD-based system is higher than that of the Hybrid one, which in turn is better than a completely HDD-based setup.

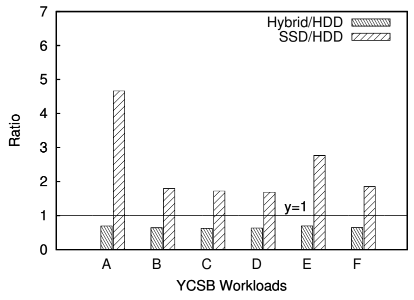
Workload A is an update heavy workload and, hence, the throughputs are lower in comparison to the other workloads. This high variance in overall throughput is in accordance with the asymmetric read/write performance of flash SSDs. Throughputs for workloads having higher percentage of reads are larger in comparison to



(a) Throughput ratio

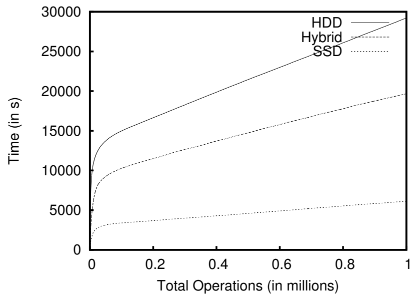


(b) Cost ratio

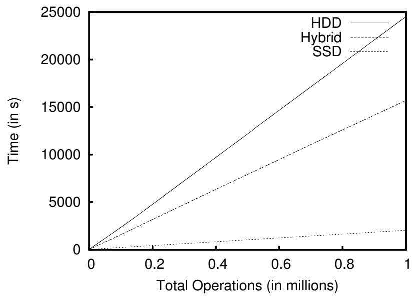


(c) Cost per throughput ratio

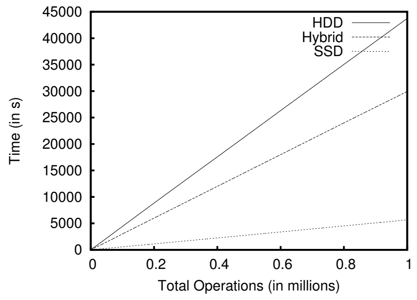
Figure 4: Relative comparison for different setups.



(a) Workload A



(b) Workload B



(c) Workload E

Figure 5: Total time taken for YCSB workloads.

workloads having no random reads (WE) or higher percentage of random writes (WA).

4.4 Performance Ratios with respect to HDD

Figure 4 shows the different performance ratios of the Hybrid and the completely flash SSD-based systems as compared to the completely HDD-based setup. The performance metrics are throughput, cost and cost per throughput. Even if the SSD-based setup gives the highest throughput for all the workloads, the cost per throughput is worse as compared to a Hybrid setup. In fact, due to the high costs of flash SSDs, it is worse than even a fully HDD-based setup. The $y = 1$ line is shown in Figure 4 to mark the base HDD-based setup.

The throughput ratio between Hybrid and HDD setups is around 1.75 for all workloads. This leads to a lower cost per throughput ratio for the Hybrid setup. The cost per throughput ratio for Hybrid setup is below 1 (approximately 0.66 for all workloads).

The difference between cost per throughput of HDD-based and SSD-based setups is even larger for workloads A and E, thereby indicating that flash SSDs are not so suitable for update heavy workloads or workloads having no random reads. Our proposed Hybrid HBase setup exhibits the lowest cost per throughput ratio for all the workloads and can, therefore, be considered the best on this criterion.

4.5 Progressive Running Time

Figure 5 shows the progressive running time for the different workloads as more queries arrive (workloads C, D and F are not shown as they exhibit similar effects). The SSD setup always performs better than the Hybrid one which in turn outperforms the HDD setup consistently.

We next measure the effect of introducing flash SSDs for garbage collection and the CPU performance.

4.6 Garbage Collection

Figure 6 shows the behavior of Java garbage collector over the three different experimental setups. The freed memory per minute is the highest for SSD setup followed by Hybrid. However, accumulative pauses are also the highest for SSD setup. Accumulative pauses are significantly larger for workloads involving updates/inserts. Thus, memory fragmentation is highest for SSD setup which further increases if an update heavy workload or an insert heavy workload is applied. Accumulative pauses due to garbage collector are higher for HDD setup in comparison to Hybrid setup. This is due to the fact that system components on flash in a Hybrid setup requires very less frequent random writes, and hence, there is less memory fragmentation and less garbage collection time.

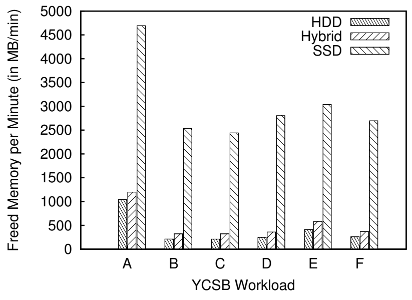
4.7 CPU Performance

Figure 7 shows the CPU utilization over the three different setups for the workloads A, B and F (others are similar to WB). CPU utilization for Hybrid setup is slightly larger than HDD setup. The CPU utilization is highest for the SSD-based setup as flash SSDs narrow the gap between I/O bandwidth and processor bandwidth. Variation of CPU utilization in WA for SSD is high as it is an update heavy workload and requires running garbage collector more frequently, thereby increasing the CPU utilization significantly.

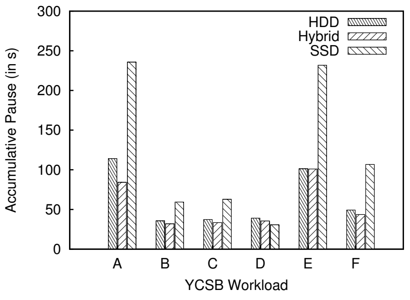
4.8 Effect of Database Size

The next set of experiments assess the impact of database size on the storage layer in the standalone system. We vary the number of records in the database, R, for $R = \{2, 4, 6, 8, 10\} \times 10^7$. Due to space limitations, we proceed only up to 6×10^7 records for a completely flash SSD-based setup. Figure 8 to Figure 13 show average latencies for all operations and overall throughputs for the six workloads A to F.

For workload A, with the increase in number of records, read latency also increases for all the setups. However, as shown in Figure

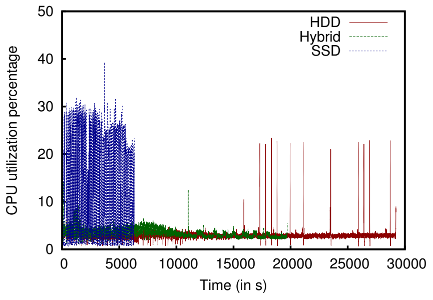


(a) Freed memory per minute

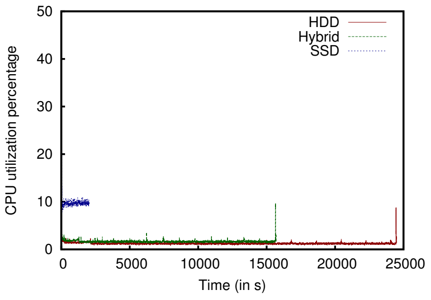


(b) Accumulative pauses

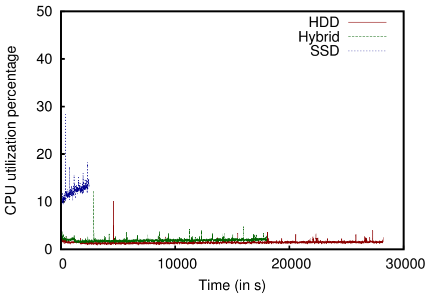
Figure 6: Effect on garbage collector over YCSB workloads.



(a) Workload A



(b) Workload B



(c) Workload F

Figure 7: Effect on CPU utilization over YCSB workloads. (Please see the soft copy version for better visualization of colors.)

8a latency increases faster for HDD setup in comparison to Hybrid setup. As number of records increase, number of regions increases as well. This leads to more accesses to *-ROOT-* and *.META.* tables which are hosted on flash SSD in a Hybrid setup. Hence, although initially with 2×10^7 records, read latencies of Hybrid and HDD setup are comparable, for larger sizes, there is a significant difference between them. Read latency of SSD is very small in comparison to other two setups as random reads are much faster on SSDs. The same behavior is shown for read latencies in workloads B, C, D and F and scan latencies in workload E.

To compare update latencies, it should be noted that while updates to a single region are sequential, those to multiple regions are random. Hence, if incoming updates/inserts are distributed across multiple regions, the random write characteristic aggravates. Update latency for workload A and B increases moderately with increasing number of records as shown in Figure 8b and Figure 9b. The update latencies for workload B is higher for all the three setups as there are only 5% update operations as compared to 50% in workload A. Since the update operations are distributed over all the regions, and the number of regions remain approximately equal for both workloads, this results in more random writes corresponding to each region for workload A. Thus, in an update heavy workload (WA), update latency for all database sizes is comparable owing to the larger sequential write characteristics.

Throughput decreases as number of records increase in all three setups for all workloads. However, as shown in Figure 8c, the change in throughput is maximum for SSD setup for workloads A, E and F. As number of regions increases, writes get more distributed. This results in smaller chunks of sequential write (random writes converted to sequential write for each region when written to new store files) for each region and larger number of such random chunks. Workload E includes insert operations and leads to many region splits. Consequently, garbage collection requirements become higher as well. Thus, the throughput drops rapidly for SSD setup for workloads A and E. The drop in throughput for the workloads B, C and D are less sharper as they are more read-heavy (Figure 9c, Figure 10b and Figure 11c).

For workloads A, E and F, the cost per throughput is the highest for SSD. With increase in number of records, it increases faster than the other two setups as shown by slope of the lines. This happens since the increase in cost is not proportional to the increase in throughput. For workloads B, C and D as well, SSD has the highest cost per throughput, but the difference with SSD is smaller as they are more read-intensive.

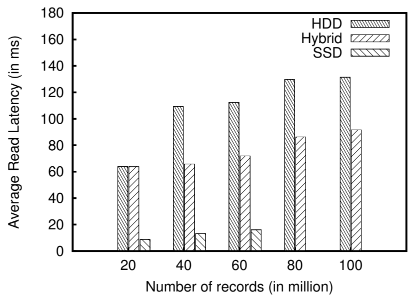
For all database sizes and all workloads, Hybrid HBase has the lowest cost per throughput. This establishes the benefits of our proposed system.

4.9 Effect of Access Pattern

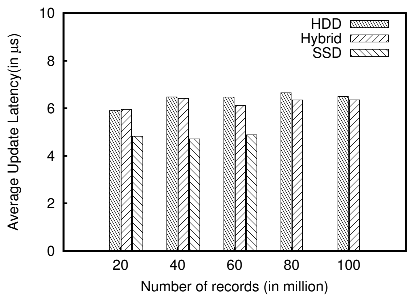
We next evaluate effect of access pattern for workloads A to F. The results are reported in Figure 14 to Figure 19.

Update latencies for the uniform access pattern are higher as compared to the other access patterns since they are distributed to a larger number of regions. To understand this better, consider the scenario where there are 5000 write operations. If these are distributed over 10 regions, then there are 10 chunks of sequential writes each containing 500 write operations. However, if these operations are distributed over 100 regions (as is more likely for a uniform access pattern), then there are 100 chunks of sequential writes each containing 50 write operations. The first will always be favorable for both HDDs and flash SSDs.

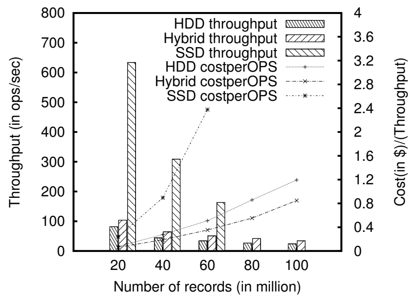
In a uniform access pattern, insert operations lead to lower number of region splits as all regions grow equally. However, in a Zipfian or latest access pattern, insert operations will happen more frequently on a few regions, thereby resulting in more frequent region splitting. Thus, in spite of having a more random write effect in



(a) Average Read Latency

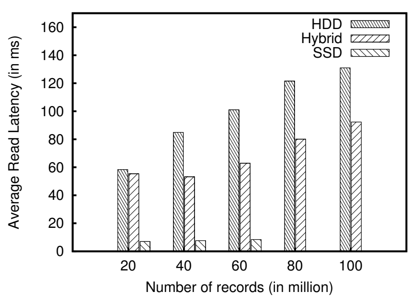


(b) Average Update Latency

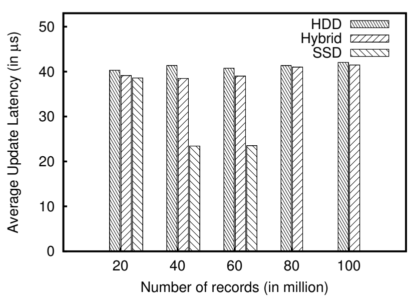


(c) Throughput

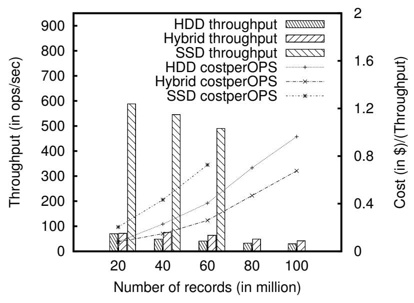
Figure 8: Effect of database size on YCSB workload A.



(a) Average Read Latency

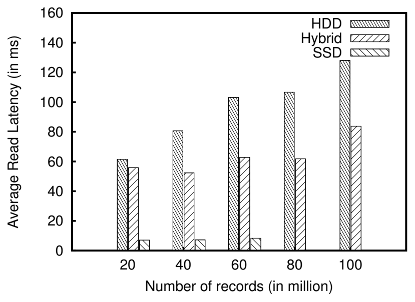


(b) Average Update Latency

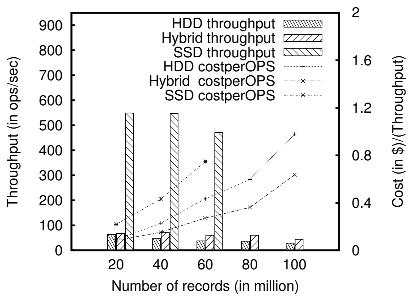


(c) Throughput

Figure 9: Effect of database size on YCSB workload B.

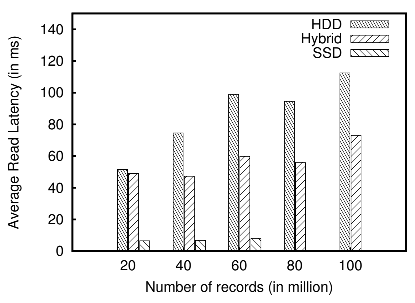


(a) Average Read Latency

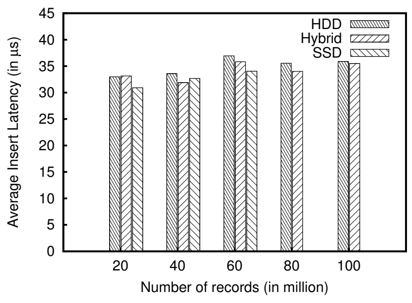


(b) Throughput

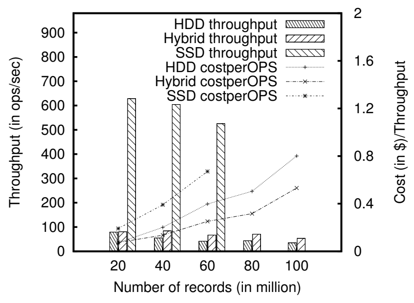
Figure 10: Effect of database size on YCSB workload C.



(a) Average Read Latency

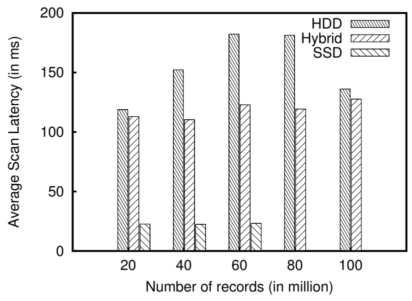


(b) Average Insert Latency

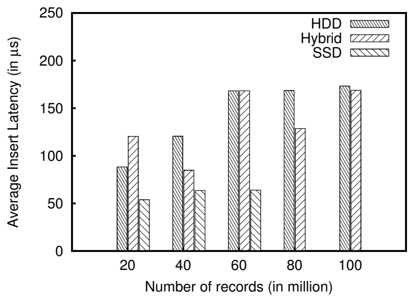


(c) Throughput

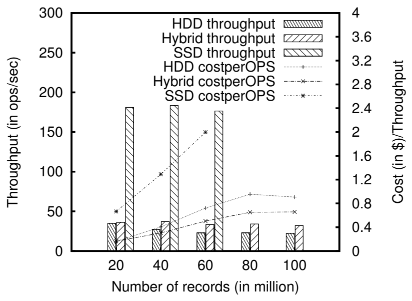
Figure 11: Effect of database size on YCSB workload D.



(a) Average Scan Latency

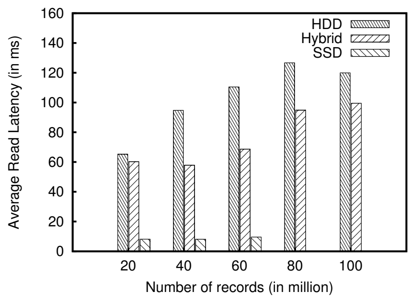


(b) Average Insert Latency

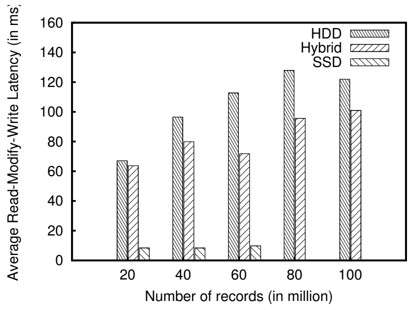


(c) Throughput

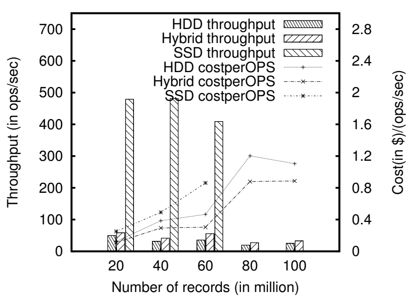
Figure 12: Effect of database size on YCSB workload E.



(a) Average Read Latency

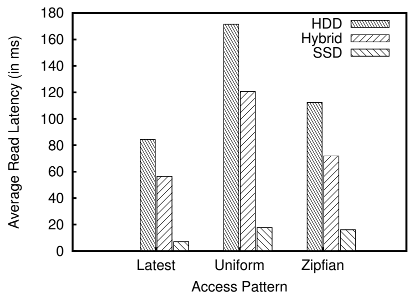


(b) Average Read-Modify-Write Latency

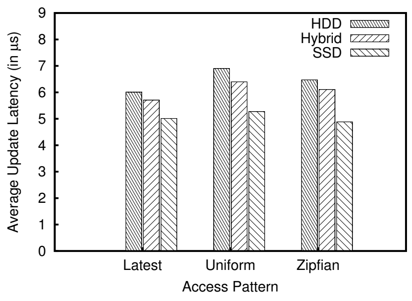


(c) Throughput

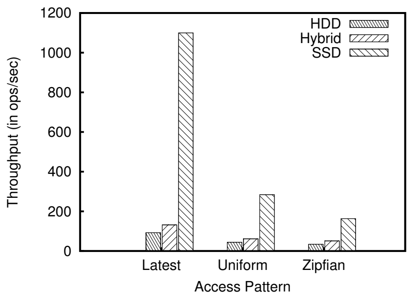
Figure 13: Effect of database size on YCSB workload F.



(a) Average Read Latency

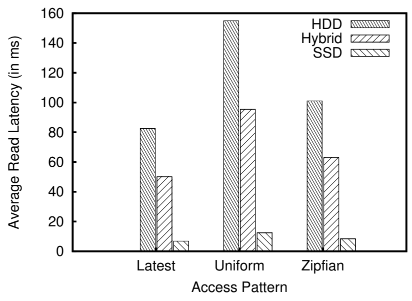


(b) Average Update Latency

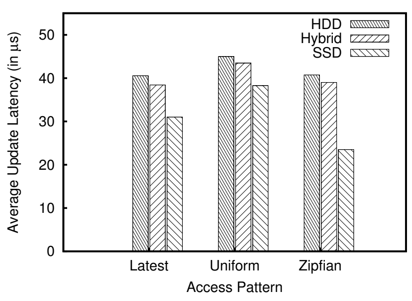


(c) Throughput

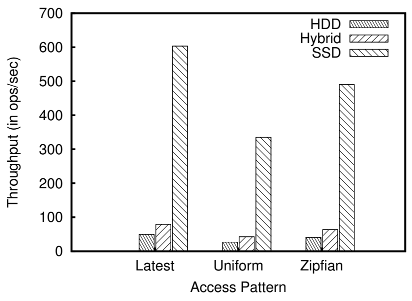
Figure 14: Effect of access pattern on operation combination of YCSB workload A.



(a) Average Read Latency

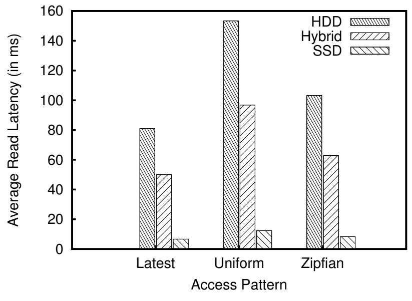


(b) Average Update Latency

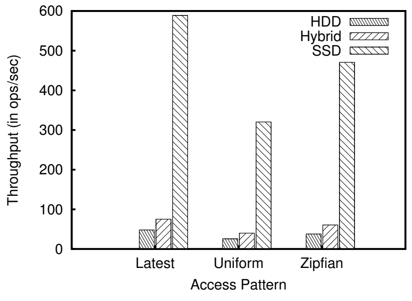


(c) Throughput

Figure 15: Effect of access pattern on operation combination of YCSB workload B.

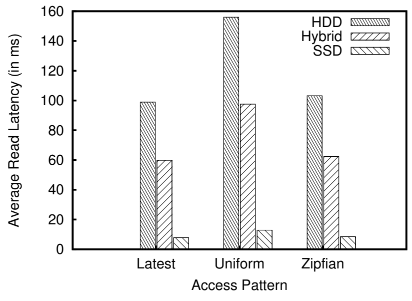


(a) Average Read Latency

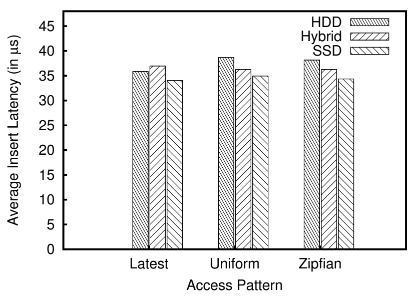


(b) Throughput

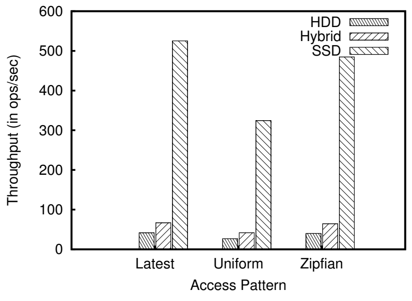
Figure 16: Effect of access pattern on operation combination of YCSB workload C.



(a) Average Read Latency

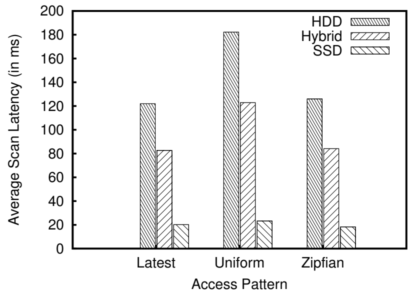


(b) Average Insert Latency

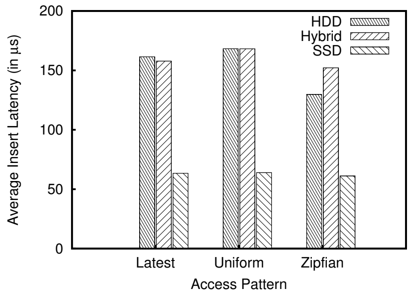


(c) Throughput

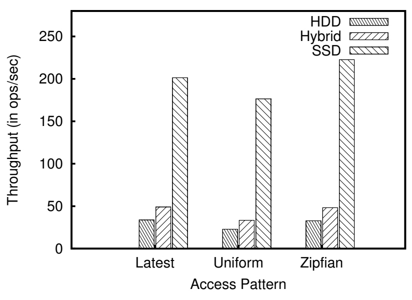
Figure 17: Effect of access pattern on operation combination of YCSB workload D.



(a) Average Scan Latency

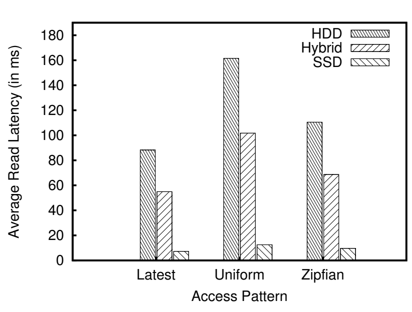


(b) Average Insert Latency

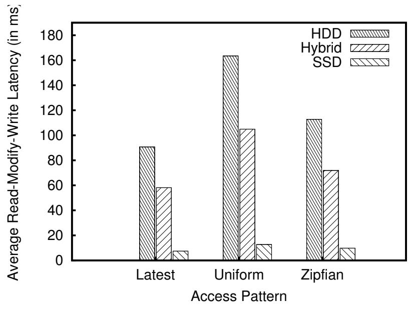


(c) Throughput

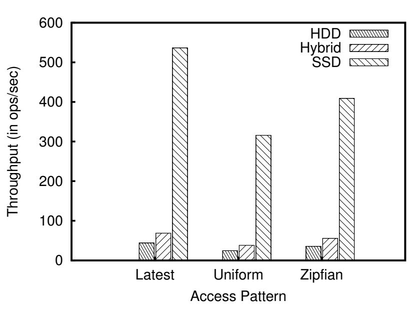
Figure 18: Effect of access pattern on operation combination of YCSB workload E.



(a) Average Read Latency



(b) Average Read-Modify-Write Latency



(c) Throughput

Figure 19: Effect of access pattern on operation combination of YCSB workload F.

update access pattern, insert latencies for all access patterns are almost similar, with only slightly higher values for uniform access pattern for all the three setups. If fewer regions are accessed more frequently for read operations, then read latency decreases due to lower cache miss. So, both read (in workloads A, B, C, D and F) and scan (in workload E) latencies are lower for Zipfian and latest access patterns in comparison to uniform. Read-modify-write operation in workload F involves a read and an update operation, and hence, has a higher latency for uniform access pattern.

Hence, throughput of uniform access pattern is the lowest for all workloads. Also, similar to previous analyses, SSD setup provides maximum throughput followed by Hybrid setup for all the tested access patterns and workloads.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we analyzed the feasibility of introducing flash SSD drives for large column store systems such as HBase. Since hosting the entire database on flash SSDs is infeasible due to its large costs, we chose only the system components. We did a thorough qualitative and quantitative assessment (by using the standard YCSB benchmark workloads) of the effects of hosting the four major system components of HBase on flash SSDs.

While a complete SSD-based solution exhibited the best throughput, and a complete HDD-based setup had the least cost, our proposed Hybrid HBase achieved the best performance in terms of cost per throughput. It was shown to be better by almost 33% than the complete HDD setup.

In future, it would be useful to assess the effects of flash specific file systems, if any. Also, we plan to extend our system to a truly distributed setup where network latencies can play an important role. Finally, it needs to be explored whether storing some data components on the flash SSD instead of the HDD can improve the cost per throughput ratio even further, and whether such a setup can be tuned automatically according to the workload.

ACKNOWLEDGMENTS

We thank NetApp Corporation, India for partly supporting this work through grant number NETAPP/CS/20110061.

6. REFERENCES

[1] D. J. Abadi. Columnstores for wide and sparse data. In *CIDR*, pages 292–297, 2007.

[2] M. Athanassoulis, A. Ailamaki, S. Chen, P. B. Gibbons, and R. Stoica. Flash in a DBMS: Where and how? *IEEE Data Engg. Bull.*, 33(4):28–34, 2010.

[3] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Grube. Bigtable: A distributed storage system for structured data. In *OSDI*, pages 205–218, 2006.

[4] S. Chen. FlashLogging: Exploiting flash devices for synchronous logging performance. In *SIGMOD*, pages 73–86, 2009.

[5] B. F. Cooper, A. Silberstein, E. Tam, R. Ramkrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *SoCC*, pages 143–154, 2010.

[6] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *SOSP*, pages 205–220, 2007.

[7] M. Du, Y. Zaho, and J. Le. Using flash memory as storage for read-intensive database. In *First Int. Workshop on Database Technology and Applications*, 2009.

[8] L. George, editor. *HBase – The Definitive Guide: Random Access to Your Planet-Size Data*. O'Reilly, 2011.

[9] G. Graefe, S. Harizopoulos, H. A. Kuno, M. A. Shah, D. Tsirogiannis, and J. L. Wiener. Designing database operators for flash-enabled memory hierarchies. *IEEE Data Engg. Bull.*, 33(4):21–27, 2010.

[10] M. G. Khatib, B.-J. van der Zwaag, P. H. Hartel, and G. J. M. Smit. Interposing flash between disk and dram to save energy for streaming workloads. In *ESTImedia*, pages 7–12, 2007.

[11] G. J. Kim, S. C. Baek, H. S. Lee, H. D. Lee, , and M. J. Joe. LGeDBMS: A small DBMS for embedded systems. In *VLDB*, pages 1255–1258, 2006.

[12] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam. HybridStore: A cost-efficient, high-performance storage system combining SSDs and HDDs. In *MASCOTS*, pages 227–236, 2011.

[13] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *Operating Systems Review*, 44(2):35–40, 2010.

[14] S. W. Lee, B. Moon, and C. Park. Advances in flash memory SSD technology for enterprise database applications. In *SIGMOD*, pages 863–870, 2009.

[15] S. W. Lee, B. Moon, C. Park, J. M. Kim, and S. W. Kim. A case for flash memory SSD in enterprise database applications. In *SIGMOD*, pages 1075–1086, 2008.

[16] Y. Li, S. T. On, J. Xu, B. Choi, and H. Hu. DigestJoin: Exploiting fast random reads for flash-based joins. In *Mobile Data Management*, pages 152–161, 2009.

[17] T. Lipcon. Avoiding full GCs in HBase with memstore-local allocation buffers. http://www.cloudera.com/blog, February 2011.

[18] A. One. YAFFS: Yet Another Flash File System. http://www.yaffs.net/.

[19] P. E. O'Neil, E. Cheng, D. Gawlick, and E. J. O'Neil. The log-structured merge-tree (LSM-tree). *Acta Inf.*, 33(4):351–385, 1996.

[20] S. Pelley, T. F. Wenisch, and K. LeFevre. Do query optimizers need to be SSD-aware? In *Second Int. Workshop on Accelerating Data Management Systems using Modern Processor and Storage Architectures*, 2011.

[21] M. Polte, J. Simsa, and G. Gibson. Comparing performance of solid state devices and mechanical disks. In *3rd Petascale Data Storage Workshop, Supercomputer*, 2008.

[22] M. Polte, J. Simsa, and G. Gibson. Enabling enterprise solid state disks performance. In *Workshop on Integrating Solid-state Memory into the Storage Hierarchy*, March 2009.

[23] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log structured file system. *ACM Trans. on Comp. Sys.*, 10(1):26–52, 1992.

[24] R. P. Spillane, P. J. Shetty, E. Zadok, S. Dixit, , and S. Archak. An eficient multi-tier tablet server storage architecture. In *SoCC*, pages 1–14, 2011.

[25] D. Tsirogiannis, S. Harizopoulos, M. A. Shah, J. L. Wiener, and G. Graefe. Query processing techniques for solid state drives. In *SIGMOD*, pages 59–72, 2009.

Entity Ranking and Relationship Queries Using an Extended Graph Model

Ankur Agrawal
IIT Bombay
ankuragrawal.iitb@gmail.com

S. Sudarshan
IIT Bombay
sudarsha@cse.iitb.ac.in

Ajitav Sahoo
IIT Bombay
ajitavsahoo@gmail.com

Adil Anis Sandalwala
IIT Bombay
sandalwalaadil@gmail.com

Prashant Jaiswal
IIT Bombay
prash.jai@gmail.com

ABSTRACT

There is a large amount of textual data on the Web and in Wikipedia, where mentions of entities (such as Gandhi) are annotated with a link to the disambiguated entity (such as M. K. Gandhi). Such annotation may have been done manually (as in Wikipedia) or can be done using named entity recognition/disambiguation techniques. Such an annotated corpus allows queries to return entities, instead of documents. Entity ranking queries retrieve entities that are related to keywords in the query and belong to a given type/category specified in the query; entity ranking has been an active area of research in the past few years. More recently, there have been extensions to allow entity-relationship queries, which allow specification of multiple sets of entities as well as relationships between them.

In this paper we address the problem of entity ranking ("near") queries and entity-relationship queries on the Wikipedia corpus. We first present an extended graph model which combines the power of graph models used earlier for structured/semi-structured data, with information from textual data. Based on this model, we show how to specify entity and entity-relationship queries, and defined scoring methods for ranking answers. Finally, we provide efficient algorithms for answering such queries, exploiting a space efficient in-memory graph structure. A performance comparison with the ERQ system proposed earlier shows significant improvement in answer quality for most queries, while also handling a much larger set of entity types.

1. INTRODUCTION

Over the last decade, there has been a lot of work on keyword search over structured and semi-structured data. Some of this body of work focuses on finding a closely connected set of data items containing specified keywords, for example [4, 10, 1, 9]. In contrast ObjectRank [2] extended the idea of PageRank to compute keyword specific ranks for objects in a connected graph. A similar idea of near queries was also mentioned briefly in [11]. All the above work focused primarily on structured data.

The 18th International Conference on Management of Data (COMAD),
14th-16th Dec, 2012 at Pune, India.

In recent years, search over annotated text data has received increasing attention. This work is motivated in part by the availability of annotated text in Wikipedia, and by the availability of text annotators for named entity recognition/disambiguation, such as [13, 19], which can work on web scale data. Such annotations add semantic links to text, identifying mentions of entities in text, and organizing the entities into a type or category hierarchy. For example, the occurrence of the words "Kleinberg" in text may be identified as a mention of the person entity "Jon Kleinberg".

Suppose the annotation on a text corpus have identified occurrences of person entities (amongst other types of entities). We can then run queries such as "find persons near Web search"; the basic idea is to find mentions of entities of type person close to the words Web and search, and aggregate over multiple such occurrences to rank persons in terms of their proximity to the words web and search. Work in this area includes [5, 6, 8] and [7]; see Section 6 for more details.

In general, entity ranking involves finding specific entities as answers to queries. The user submits the search keywords and also the target type of the desired answers. (We use the words type and category interchangeably, since both terms have been widely used in prior work.) Some examples of such queries as obtained from the INEX 2008 track are: "Find a list of musicians who appeared in at least one of the Blues Brothers movies", and "Find a list of the state capitals of the United States of America".

Wikipedia is often used as the source of entities, and the YAGO category hierarchy [18] (which provides a cleaned up version of Wikipedia categories combined with the WordNet ontology) is used to associate entities with a hierarchy of categories. Several systems, such as Yago, also extract relationships from unstructured information and represent them, for example, using RDF or even relational schemas. Structured queries are then run on the structured data, by systems such as Naga [12], [3] and [17]. However, the number of extracted relationships are limited, and the integration of unstructured and structured information is limited. See Section 6 for more details.

The ERQ system [14, 15] has worked on more complex queries called entity-relationship queries, that can look for relationships between entities. Queries can specify entities in a manner similar to entity ranking queries, but additionally specify desired relationships through keywords. As an example from [15], a query can ask for "persons related to Stanford who have founded companies in silicon valley"; more formally the query asks for "person entities near Stanford that are related to company entities near silicon valley by the term founded".

The systems mentioned above exploit entity annotations, but do not exploit the graph structure of the underlying data. For example, they cannot answer a query of the form "find universities near Nobel prize" unless there are mentions of the term Nobel prize near the university name. If person entities related to Nobel prize, are also related to a university entity, we would consider the university to be related to Nobel prize. Such transfer of prestige does occur in graph based systems such as Object Rank [2] and BANKS [11], but those systems do not support nodes containing annotated text. Our goal is to have a unified model that handles both graph information and annotated textual data.

As a first attempt to address the issue, we treated the Wikipedia corpus as a graph, with documents as nodes and inter-page links as edges, and ran the near query implementation of [11] on the graph. However, the results were very disappointing; the main reasons were (a) the graph is very densely connected and (b) it makes no sense to consider a link at the end of a long Wikipedia page to be related to a word that occurs early in the page.

To address the above problem, we introduce the notion of a graph where nodes contain words and edges, occurring at specified offsets. When we traverse the graph to answer a query, we take the offsets into account, in a way that we describe later in the paper. This extended graph model is well suited to Wikipedia data, to annotated Web pages, as well as to traditional structured data, and can be used in systems that integrate different types of data.

We then show how to use the extended graph model to define scoring models for entity and entity-relationship queries, and to derive efficient algorithms for answering such queries.

The contributions of this paper are as follows:

1. We present (in Section 2) a new graph model that allows nodes to contain terms as well as links at specified offsets. This model combines the best features of the graph model, and the document models, both of which have been widely used in the past.

2. We present (in Section 3) new methods for scoring answers to near queries taking the new graph model into account.

   Unlike earlier work on entity ranking and entity-relationship queries, our model does not require the user to provide a precise specification of the desired type of the results; instead, we allow type-keywords to describe the desired type. Answers are scored based on how well the type matches the given type-keywords, and the entity matches the remaining keywords.

3. We then present (in Section 4) a scoring model for entity-relationship queries, again based on the extended graph model.

   We also present efficient algorithms for answering entity-relationship queries in the above graph model.

4. We present (in Section 5) several optimizations improve result quality.

5. We present a performance study (in Section 7) which shows that our techniques give good result quality, outperforming [15] on most queries.

2. DATA MODEL

We now describe our extended graph model, and then outline how semi-structured datasets from Wikipedia and YAGO [18] can be represented in the extended graph model.

2.1 Extended Graph Model

The basic data model we use is a labelled directed multigraph $G = (V, E)$, where V is a set of vertices and E a multiset of edges. Our multigraph model has two further extensions to better handle documents.

1. Vertices can have an associated text description, modeled as a document; such vertices have an associated set of *(term, offset)* pairs. The offset denotes the relative position of the term from the start of the document. Vertices that do not represent documents can still have text descriptions, with all terms assumed to be at offset 0.

   Vertices can have associated labels; for example, when modeling Wikipedia, these labels can be used to distinguish regular entity nodes from category nodes. Vertices can also store other information, for example a node prestige may be associated with each node.

2. Edges are directed. Edges can represent a hyperlink from one document to another; each such edge $e = v1 \rightarrow v2$ has an associated offset $e.offset$ which is an offset within the document represented by $v1$ where the hyperlink occurs. There can be multiple edges from one vertex to another, at different offsets, which is why we use a multigraph model. Edges that do not represent hyperlinks are assumed to have an offset of 0.

   Edges can also have associated labels; for example, when modeling Wikipedia data, edge labels can be used to distinguish edges linking a node to its category, from edges linking a node to a non-category node. Edges can also have an associated edge weight.

We call the above graph model as the *extended graph model.*

The extended graph model only stores nodes and edges with offset information. The mapping from terms to nodes (including offset information for term occurrences) is stored separately, in a full text Lucene index. The term frequency (TF) of each term in a document can also be stored in Lucene; for example, terms in a document title can be given a higher TF. The node prestige of a node can also be used to boost the score of the corresponding document in Lucene.

2.2 Representing Wikipedia Data

In Wikipedia, every entity is stored as a separate document (a Wikipedia page) also called articles. Wikipedia articles are all linked or cross-referenced. These articles are categorized according to the type of entity it represents. Wikipedia provides us with category types, into which an author could categorize the pages.

In our model, each Wikipedia page/document represents an entity, which is the basic unit of our search and thus, it is represented by a node in the graph. There are two types of nodes in our model:

- *category nodes* (representing Wikipedia categories)
- *entity nodes* (representing all other Wikipedia pages)

Each vertex has a label identifying whether it is a category vertex or a entity vertex. In addition, each vertex has a separate label denoting its page-rank, pre-computed as described later. If we integrate other Web pages into our graph, we could use a new node type, *web-page* node, to represent such Web pages,

Labels are also associated with the edges to identify the edge type; the different types of edges in the graph are as follows.

1. Document to entity edges, which link from a document to entities referenced in the document. Each entity has an associated document in Wikipedia. The offset associated with such an edge is the token offset of the start of the link in the document.
2. Edges denoting the 'belongs to' relation from an entity to a category The offset of such edges is 0.
3. Edges denoting category to category hierarchy; the offset of such edges in 0.

Since, a single data graph is built for both entities and categories different parts of the graph can be traversed based on the edge type.

Edges linking entities to categories that denote the "belongs to" relationship are treated specially for the purpose of ranking.

As in [4, 11], the node prestige of a node is a measure of its importance disregarding query keywords, and is computed using a biased PageRank computation with edge weights, as described in [11], with teleport probability of 0.3. Offset information is ignored, and all edges in the original graph are treated as being of unit weight.

As in [11], for each directed edge $u \rightarrow v$ in the original graph, we introduce a reverse edge $v \rightarrow u$, if such an edge is not already present. Each reverse edge is assumed to be at offset 0, and its weight is defined as the indegree of v.

The Wikipedia category hierarchy has a number of problems, such as cycles, and improper nesting of categories. For example, Jerry Yang, the founder of Yahoo! is in the category Yahoo!, and thus indirectly (after a few more levels in the hierarchy) under the category Companies. Based on the hierarchy, we expect each entity to belong to higher level categories also. However, we would certainly not expect Jerry Yang to be categorized as a company.

To avoid these problems we used the category hierarchy of the YAGO ontology [18]. YAGO includes all Wikipedia entities, as well as conceptual categories from Wikipedia, but replaces the Wikipedia category hierarchy by the WordNet hierarchy, suitably integrated with the Wikipedia categories (which now form the leaf level of the category hierarchy). This not only improved the quality of results, linking entities only to relevant categories in most cases, but also reduced the execution time significantly.

3. NEAR QUERIES

In this section, we first describe our model for *near queries*, and then describe how answers are scored using our extended graph model.

3.1 Near Query Model

A *near query* q can be specified as

find C **near** (K)

Where C is one or more keywords specifying the target entity type for the answer, and K is a set of keywords; phrases enclosed in double quotes can also be used in place of keywords to ensure that the keywords appear together in the order specified.

Example. Consider a user searching for the list of movies in which actor Robert De Niro has played a part and is directed by famous Hollywood director Martin Scorsese. The **near query** formulation of this query will be:

find films **near** (directed "martin scorsese" "robert de niro")

Here the keyword *films* gives the type information C, and the set K is equal to {directed, martin scorsese, robert de niro}.

Near queries using the above syntax were supported in the BANKS system [11]. However, as mentioned earlier, when we attempted to use the BANKS near query model on the Wikipedia corpus, with Wikipedia pages modeled as nodes, the performance was very poor; the reason is that nodes have many keywords and many links, and a keyword occurring early in a page often has little connection to a link occurring late in the page. In Section 3.2 we describe how to score answers based on proximity of keywords to links or entity mentions.

We use the following terminology in the rest of the paper:

- *categoryKeywordList*: Keyword (or set of keywords) C before the meta-word *near* which specifies the target categories (entity types).
- *nearKeywordList*: The set of keywords following the meta-word *near*. Each keyword is separated by space within the parenthesis. Keywords within quotes are considered as phrases and as a result, single keywords.
- *nearKeywordOriginSet*: The document pages that contain the keywords in the *nearKeywordList*.
- *relevantCategorySet*: The set of categories relevant to *categoryKeywordList*.

We could use either of the following alternatives to decide which documents form the nearKeywordOriginSet:

- AND semantics: Every document in the *nearKeywordOriginSet* must contain all the keywords in *nearKeywordList*.
- OR semantics: Every document in the *nearKeywordOriginSet* must contain at least one keyword from the *nearKeywordList*.

In our implementation we use the default scoring mechanism of Lucene, which corresponds to the OR semantics.

3.2 Scoring model

We now see how to score answers to near queries, using the idea of activation spreading, as well as the relevance of a category to the category keywords in the query. Our technique extends the spreading activation technique used for near queries in [11], by taking the proximity between keywords and links (calculated using offset values) into account. Our scoring models have a number of parameters; default values are specified for some of the parameters when they are introduced, but the values used in our experiments are given later, in Section 7.3.

3.2.1 Activation Spreading

As described in [11], activation spreading is initiated from the nodes containing keywords, and spreads activation to neighboring nodes. The following are the key features: (a) The initial activation from a given keyword is spread to nodes containing that keyword, in proportion to the node prestige (PageRank) of each such node. Nodes that receive the maximum activation form the results of the near query. (b) Each node retains part of its incoming activation, and spreads the remaining to its neighbors; the fraction spread to each neighbor is inversely proportional the weight of the directed edge from the node to its neighbor. (c) Activation received from multiple neighbors is combined using a combining function. Activation spreading continues until the amount spread falls below a specified threshold.

We now describe how the above scheme is modified in our context.

3.2.1.1 Initial Activation.

In our context, activation spreading starts from nodes representing the documents which contain the keywords. The initial hit set for query keywords is obtained using the searcher available in Lucene. Lucene also returns the score of the documents that are obtained as hits during the search.

The initial activation of a node is a combination of of the relevance of the node to the keywords, given by the Lucene score for the node, and the node prestige of the node. The *initialActivation* value for each node is calculated from these two scores by combining them either additively:

$$\mathit{NodePrestige} * \alpha + \mathit{LuceneScore} * (1 - \alpha) \tag{1}$$

or multiplicatively:

$$[\mathit{LuceneScore}^{\alpha}] * [\mathit{NodePrestige}^{(1-\alpha)}] \tag{2}$$

Here α is a distribution factor that can be tuned to give more weight to the desired score. By default we use multiplicative combination with $\alpha = 0.5$.

Note that the above model is a little different from the near query model of [11]; that model allowed each near keyword to appear in a different tuple, and spread activation separately for each near keyword. The activation scores were combined across multiple keywords either multiplicatively (for the AND semantics) or additively (for the OR semantics). In the context of search on Wikipedia and other document collections, it makes more sense to compute the initial activation across all keywords, and then spread activation only once.

3.2.1.2 Proximity and Spreading of Activation.

When spreading activation from a node, an attenuation factor μ is used. Every node spreads a fraction $1 - \mu$ of its activation to its neighbors and retains the remaining μ fraction for itself. By default, we set $\mu = 0.75$. As in BANKS, the fraction of activation spread to each neighbor depends on the edge weights. However, the spreading of initial activation is special cased. The fraction of the initial activation spread to each outlink depends on the proximity of the outlinks to the near keywords. Intuitively, if a keyword and a link to an entity occur in proximity in a document, we believe that the entity is related to the keyword; the closer the occurrences, the higher is the estimate of relevance of the entity to the keyword. We use this idea to define the amount of activation transferred to each of the entities linked with the document.

The position offset of each term of a document is stored with the index. And the offset information for every link in a document is stored in the graph during pre-processing phase. This offset is calculated with respect to the start of the document. The amount of activation spread to the entity pointed to by the link is proportional to the distance between the link and the query keyword in the document.

The function to calculate the proximity of a link with respect to a keyword must be such that its value degrades as the distance between the link and the keyword increases.

Formally, if a word w occurs at position i, and a link to an entity at position j, then if the position j is closer to i, the propagated activation for word w at that position would be larger than the propagated activation at a position farther away. The issue of how the activation should decay with distance is studied in [16]. We use the Gaussian kernel function to calculate the proximity score.

$$k(i, j) = exp[\frac{-(i-j)^2}{2\sigma^2}]$$

The initial activation associated with a node in nearKeywordOriginSet is spread to the outlinks of the node in proportion to proximity (using the formula defined above) based on the distance between the outlink and the nearest occurrence of the near keyword; with multiple keywords, we take the distance as the minimum, across all keywords, of the distance as above.

3.2.2 Category Relevance

The answers to a keyword query must satisfy the target type information specified in the query. In the near query model, a user specifies the target type for the answers by providing relevant keywords. In the context of near queries, this target type specifies one or more categories, and the result entity must belong to one of these categories. Each category has a category relevance score, which is used in entity ranking.

The categories are indexed separately, as documents, and the categoryKeywordList specified in the query is used to retrieve relevant categories; we call the set of categories returned as the *relevantCategorySet*. We use the relevance score that Lucene returns for each category as the relevance of that category.

To calculate relevance score of an entity, the set of categories to which this entity belongs is retrieved. It is then checked if any of these categories belongs to *relevantCategorySet* and the maximum of the Lucene scores of such categories is taken as the category relevance of that entity.

3.2.3 Combining Activation and Category-Relevance Scores

After spreading of activation, the result of activation spreading is stored in a priority heap ResultHeap. To get the final score *score* of each entity, the activation score *actScore* and the the category relevance score *relScore* of each node in ResultHeap are combined additively as follows:

$$score(e) = actScore(e) * \eta + relScore(e) * (1 - \eta) \quad (3)$$

The parameter η denotes the weight given to the score. Entities in the result are sorted by their scores $score(e)$, and output in descending order.

3.3 Discussion

Our scoring model for near queries spreads activation from entities to other entities that are referenced in the Wikipedia page of the entity (only links in or before the infobox are considered, since Wikipedia pages often have less relevant links later in the document).

For example, if we search for *Universities near "web search"*, we may find many references to a person working on web search techniques near keywords "web search". Spreading activation from such person entities can then give us a university as an answer.

Earlier systems such as [5, 8, 7] and [15] (described in more detail in Section 6) cannot do this, since they only look for co-occurrences of entities and keywords to determine their association.

4. PROCESSING ENTITY-RELATIONSHIP QUERIES

In this section, we focus on issues involved in answering entity-relationship queries. The query model we use is basically the same as that described in [14, 15], but we use a different scoring system, as well as a different system design and implementation to solve such queries.

In our formulation of the entity-relationship queries, as in [15], we have a list of *entity variables*. Unlike in [15], each entity variable is associated with a list of keywords specifying the category of the desired entities called *categoryKeywordList*, and these category keywords are used to identify one or more categories to be considered for the entity variable.

Each entity variable can be associated with zero or more predicates. There are two kinds of predicates in an entity-relationship query :

- **Selection Predicate :** A selection predicate consists of an entity variable and a list of keywords specifying the criterion on the selection of entities. We call the list of keywords as the *NearKeywordList*.
- **Relation Predicate :** A relation predicate consists of two or more entity variables and a list of keywords specifying the relationship between the entities described by these variables.

As an example consider the following query from [15]: *"Find companies and their founders, where the companies are in Silicon Valley and founders are Stanford graduates"*. Simple entity ranking systems are not adequate for such complex information needs. Li et al. [15] provide a solution to this problem, by designing an entity-centric structured query mechanism called *entity-relationship queries*. The above query expressed in the language of [15] is as follows:

select X, Y
from person X, companies Y
where X:[Stanford graduate]
    and Y:["Silicon Valley"]
    and X,Y: [founder]

In the above query, X and Y are entity variables, bound to specific entity types, while the keywords act as predicates.

The above query can be expressed in our syntax as follows:

find person(x) **near** (Stanford graduate) **and**
    company(y) **near** ("Silicon Valley")
such that x,y **near** (founder)

In this query, there are two entity *variables* named x and y. The *categoryKeywordList* for variable x contains the word *"person"* and for variable y, it contains the word *"company"*. Variable x has a *selection predicate* consisting of keywords *"Stanford"* and *"graduate"* while variable y has a *selection predicate* consisting of keyword *"Silicon Valley"*. The query also has a *relation predicate* on variables x and y consisting of keyword *"founder"*.

As in ERQ [15], an entity variable can have more than one selection predicates. For example

find person (x) **near** ("Turing Award")
    and near (IBM)

If we had instead used **near** ("Turing award", IBM), we would only get entities mentioned near co-occurrences of Turing Award and IBM. In contrast, by using separate selection predicates, the set of documents that establish that a person is associated with "Turing Award" can be different from the set of documents that establish that the person is associated with IBM.

4.1 Scoring ERQ Answers

Scoring and ranking of the results is an important task. The important concepts involved in ranking the entity search results are:

- **Proximity**: Entities and keywords should be placed close to each other in the text. Intuitively, the closer they are to each other, the more likely is their association with each other.
- **Relevance to category**: As the category itself is specified in the form of keywords, there is uncertainty involved regarding the relevance of an entity to the specified category keywords.
- **Number of Evidences**: The more number of times a set of entities appears with the keywords in the text, the more likely is their association.

First, we score each answer entity tuple for each predicate separately. Finally while merging the single predicate results, we calculate the aggregate score for each answer tuple by taking the product of the single predicate scores for the entities involved.

4.1.1 Selection Predicate Scoring

A selection predicate in an entity-relationship query is basically a near query, which we saw in Section 3. To compute score of an answer entity e on a selection predicate p, we use the scoring model for near queries described in Section 3. We combine the activation score *actScore* and the category relevance score *relScore* using the additive combination:

$$score_p(e) = actScore(e) * \eta + relScore(e) * (1 - \eta)$$

The combined score is a normalized score and the value is always between 0 and 1.

If there is more than one selection predicates over the same variable, we use the following formula, where $p_1, p_2, \ldots p_n$ denote the selection predicates on a single entity variable.

$$Score_{p_1,p_2,\ldots,p_n}(e) = (\Pi_{i \in 1 \ldots n} actScore_{p_i}(e)) * \eta + relScore(e) * (1 - \eta)$$

4.1.2 Relation Predicate Scoring

Consider a relation predicate answer tuple $< e_1, e_2, ..., e_n >$, and the set of occurrences O of the entities in the answer tuple and the keywords corresponding to the predicate appearing together in the text. We calculate the score for the relation predicate p as:

$$score_p(< e_1, e_2, ..., e_n >) = \sum_{o \in O} exp[\frac{-(TokenSpan(o))^2}{2\sigma^2}]$$

where $TokenSpan(o)$ is the number of tokens present in the minimal scope in o covering all the entities and keywords. λ is an input parameter specifying the threshold for the maximum allowed value of $TokenSpan$ and all occurrences beyond this threshold are ignored.

4.1.3 Aggregating Single Predicate Scores

After computing single predicate scores for each predicate result, we finally merge the results and calculate the aggregate score for the final answer tuples. The aggregate score *aggScore* is calculated as :

$$aggScore = \prod_{p \in selPreds} score_p * \prod_{p \in relPreds} score_p^{\gamma}$$

where *selPreds* and *relPreds* denote the selection and relation predicates, and γ is an input parameter controlling the weightage given to the relation predicate scores.

4.2 Query Evaluation Algorithm

Given an entity-relationship query, our approach is to first evaluate all the selection predicates individually to find the list of entities for each entity variable involved in the query. We then use these entity lists to evaluate the relation predicates to find tuples of related entities. Finally we take a join of the individual predicate result list on entities for same entity variable. In the process, we also collect offset information to finally score the answer tuples and rank them accordingly. We look at the steps involved in evaluating an entity-relationship query in the following sections.

4.3 Evaluating Selection Predicates

A selection predicate in Entity-Relationship Query is exactly a near query. So we directly use the near query evaluation algorithm described in Section 3 to get the list of answer entities for each entity variable, along with their scores.

After this step, we will have a list of <*entity, score*> pairs for each variable. For our example query, the lists would be:
variable x : <Scott_McNealy, 1.0>, <Ken_Kesey, 0.9973>, <John_Steinbeck, 0.9946>, ...
variable y : <Microsoft, 1.0>, <Hewlett-Packard, 0.9944>, <Metro_Newspapers, 0.9942>, ...

4.4 Evaluating Relation Predicates

Relation predicates specify a relationship between two or more entities in terms of keywords. There are two alternative approaches to solve a relation predicate.

Approach 1

- Use the Lucene index to find documents containing the relation keywords, along with their offsets in the documents.
- For each Lucene hit page :
  - Find entity references near those keyword occurrences, using the outlinks from the entity pages (outlink information along with offsets is available in the extended graph representation, stored in-memory).
  - Check whether these entities belong to the selection predicate answer entity list for any of the variables involved in this relation predicate and put them in a list for the corresponding entity variable.
  - Perform a cross product of the lists for the entity variables, to get the answer tuples.
  - Note the offsets of the keywords and the entity links for score calculation.

The problem in this approach is that in most cases, the keywords specifying the relationship are very general (e.g. “join”, “found” etc.) and generate a very large number of hits. However only a small fraction of these pages contain links to at least one entity from the selection predicate answer list for each entity variable involved in this relation predicate. Thus processing each document as above causes a lot of useless processing.

We solve this problem using Approach 2 described below.

Approach 2. The result of a single relation predicate, taking into account selection predicates on all the associated entity variables, can be computed as follows.

- Find lists of pages containing reference to at least one of the entities in the selection predicate answer list for each entity variable; this can be done using the inlinks of the corresponding entity nodes, fetched from the in-memory graph representation.
- Intersect these lists to find list of pages containing links to at least one entity from the selection predicate answer list for each entity variable.
- Intersect this list with the hit list for the relation keywords to find all such pages also containing the relation keywords.
- For each page in this list:
  - Perform a cross product of the entity lists for each entity variable present in this page to get the answer tuples.
  - Note the offsets of the keywords and the entity links for score calculation.

```
1: Inputs: List of entity variables: $eVars$,
   List of Keywords: $nKeywords$,
   Mapping of variable to Entity list: $varToEntityMap$
2: Define: $VarToPageMap$: a mapping from entity
   variables to list of pages
3: Define: $VarPageToEntityMap$: a mapping from
   <entity-variable, page> pairs to a list of entities
4: for all $v \in eVars$ do
5:   for all $entity \in varToEntityMap[v]$ do
6:     $pageSet \Leftarrow$ Find all pages pointing to $entity$
7:     for all $page \in pageSet$ do
8:       $VarToPageMap[v]$.Add($page$)
9:       $VarPageToEntityMap[v, page]$.Add($entity$)
10:    end for
11:  end for
12: end for
13: $allLinkPageList \Leftarrow \cap_{v \in eVars} VarToPageMap[v]$
    /* Computes intersection of lists*/
14: $LuceneHitArray \Leftarrow$ Find all pages which contain the
    the keywords $nKeywords$ using the Lucene Index.
15: for all $luceneHitPage \in LuceneHitArray$ do
16:   if $luceneHitPage \in allLinkPageList$ then
17:     Define $varToEntitiesMap$: a map from entity
        variables to a list of entities
18:     for all $v \in$ eVars do
19:       $entityList \Leftarrow VarPageToEntityMap[v, NodeId]$
20:       $varToEntitiesMap[v]$.Add($entityList$)
21:     end for
22:     $answerTuples \Leftarrow \times_{v \in eVars} varToEntitiesMap[v]$
        /* Compute cross product ($\times$) of entity lists;
        optimization using band join described in text*/
23:     $ResultHeap$.addAll($answerTuples$)
24:   end if
25: end for
```

Algorithm 1: Evaluating a relation predicate

An optimization of the this step is to perform a band merge of lists sorted on their offsets, to only match entity and relation keyword occurrences that are present close to each other (in terms of their offsets), instead of performing a cross product of the entity lists. This can reduce costs greatly for pages with many entity references and relation keyword occurrences.

The above intuition is formalized in Algorithm 1.

After this step, we have the list of entity-tuples with their relation predicate scores. For our example query, we will have a list like:

x,y : <(Bill_Gates, Microsoft), 0.9896>,
<(David_Filo, Yahoo!), 0.9745>,
<(Vinod_Khosla, Sun_Microsystems), 0.9257>, ...

4.5 Handling Complete Queries

If a query does not involve any relation predicate, processing is straightforward. If the entity variable in such a query has more than one selection predicate, we need to combine the results of each selection predicate; we use a simple merge join of the results.

If the query involves only one relation predicate, Algorithm 1 gives the desired final answers. In case the query has more than one relation predicate, we process each relation predicate as above, and then do an equijoin on the results of each selection predicate. Currently we do not optimize the join order, since none of our benchmark queries has more than 2 relation predicates, but this could be a topic of future work.

As an optimization, if a query has an entity variable with more than one selection predicate, as well as a relation predicate involving the same entity variable, we can avoid the join of the selection predicate results; instead, when we process the relation keyword we get a list of neighboring entities for each keyword occurrence, and look up such entities in the result entity lists for each of the selection predicates on that entity variable.

5. HEURISTIC OPTIMIZATIONS

We now describe a few heuristics aimed at improving the scoring of results. The effect of these optimizations is studied empirically in Section 7.

Using Wikipedia Infoboxes. In our initial implementation, every term in a Wikipedia article was assumed to be relevant to the entity. However, our initial experiments showed that most Wikipedia articles have a lot of terms that are not very relevant. However, the terms early in the article, in particular those that occur in the Wikipedia infoboxes, are highly relevant. We could have chosen to tailor the ranking scheme of Lucene, but instead chose to use our extended graph model to exploit this information, as follows.

When we build the graph, we assume that a self-link to the same Wikipedia entity is present near each term in the infobox, at a small offset (with default value as 5). Thus, if we find some keyword in the infobox, we add some initial activation to the entity itself. Similarly, we create self links to the Wikipedia page from terms in the first few sentences of each article; for concreteness, we use all sentences that appear before the infobox in the article, since these generally constitute a highly relevant summary of the entity.

Exploiting Wikipedia category specificity by matching near keywords. Another area of performance improvement is the specificity of Wikipedia categories. Wikipedia provides a large collection of categories, many of which are associated with very specific entities. For example, *Novels_by_Jane_Austen*, *Films_directed_by_Steven_Speilberg*, *Universities_in_Catalunya* are all Wikipedia categories.

Users are generally not aware of the presence of such categories, and would query on a higher level category, for example novels, even if they are specifically looking for novels by Jane Austen.

Thus, we look for the *near keywords* in the category titles also. If we find any category whose title contains all the *near keywords*, we judge entities belonging (directly) to the category as being more relevant to the near keywords. If such a category is a subcategory of the original query categories, the resulting entities are directly answers to the original query. But even otherwise, we wish to give extra weight to such entities for the purpose of spreading activation through entities that occur close to occurrences of the near keywords.

To handle both the above goals, we add a constant value (0.2) to the initial activation to entities directly belonging to the above category; if an entity belongs to more than one such category, its initial activation gets increased only once.

We demonstrate the effect of this feature in Section 7.

Spreading activation from articles with title containing the near keywords. Intuitively, if the title of an article contains all the near keywords, all the content in the article can be assumed to be related to the keywords with high probability. We exploit this intuition by spreading activation from such articles to its out-neighbors.

In our spreading activation mechanism, the activation decays for links farther away from the keyword occurrence. In the special case of keywords in the article title, we treat all outlinks early in the article (up to and including the infobox for the article) as closely related to the keyword, even if they are somewhat further off in terms of token offset.

We demonstrate the effect of this feature in Section 7.

6. RELATED WORK

Several systems such as ObjectRank [2], the system of [5] and Entity Search [7] have been developed which return a ranked list of entities as answers to keyword queries.

ObjectRank works on a graph model of data, with entities as nodes, and is based on a biased random walk model which is an extension of the random walk model of PageRank. Nodes also contain descriptive text, with the starting nodes of the walk being determined by which nodes contain the given keywords. The biased random walk determines the score of each entity. The near query model of BANKS, briefly mentioned in [11] uses a similar model, and has similar goals, although details vary.

Chakrabarti et al. [5] describe an entity querying system based on a model where documents have terms as well as entity mentions. Queries can specify the type of the desired entities, and keywords that they should be associated with. For example, a query may ask for "cities near Eiffel tower". The occurrence of an entity mention near the given keywords provides support for the relevance of that entity. The support for an entity is aggregated across multiple occurrences of mentions of that entity near the given keywords. Chakrabarti et al. [5] also describe a query language which allows more complex queries to be created, allowing for example entities that occur near entities retrieved by a subquery. The implementation in [5] worked on a Web scale corpus, but was limited to a small number of entity types; that limitation was subsequently removed [6].

EntityRank [8] has a similar goal, and also works on Web scale data, but allows recognition of multiple entities co-occurring with given keywords. Specifically, it allows the query to specify multiple target entity types, such as #professor, #university, along with keywords such as "database". All entities and keywords should co-occur near each other in the same document. Entity Search [7] has goals similar to that of [5], but focuses on efficient evaluation of queries by creating appropriate indices.

Users are however often interested in relationships between entities, where the keywords that select entities may occur separately from keywords that specify the desired relationships; we give an example shortly. If the relationships have been extracted already, it is possible to represent the information using a graph model such as RDF, and then a query language such as NAGA [12] can be used to execute such queries on the graph. The NAGA query language allows complex connections to be specified, and allows aggregation of evidence from multiple parts of the graph. However, a problem with this approach is that relationships have to be extracted ahead of time, and at Web scale the number of potential relationships is enormous. The YAGO dataset [18] used in the NAGA system only extracted a few tens of relationships. Other related work which considers integration of structured information and textual data includes the ESTER system [3] and Pound et al. [17]. Both these systems focus on relationships that have already been extracted, using the YAGO dataset, and thus support only a limited number of relationships. However, both these system allow queries to combine some form of textual search with the queries on structured data.

The ERQ system [14, 15] presents an alternative approach where the corpus is stored uninterpreted except for identification of entities. Relationships are specified by keywords, and found by keyword search on the corpus, in effect performing a simplified on-the-fly extraction of relationships.

ERQ uses three position-based features for ranking answers tuples. The first is *proximity* which emphasizes the fact that if the entities and keywords are close to each other in an evidence, then it is more likely to form a valid evidence. The second feature is the *ordering pattern* of entities and phrases in an evidence. The ordering patterns which appear more often are better indicators of valid evidences. The third feature is the *mutual exclusion* rule which dictates that when evidences of different entities co-occur in the same sentence, at most one colliding pattern is effective. Our scoring model takes proximity into account, but does not currently implement ordering patterns and mutual exclusion.

Although the ERQ system does not limit the number of relationships, the evaluation algorithm used in ERQ requires separate indices per entity type, and the implementation of [15] indexed only 10 selected entity types. Thus the number of queries that can be expressed is limited. In contrast, in our system, we handle all possible categories specified in Wikipedia/YAGO. To our knowledge, all the earlier systems require the answer types to be precisely specified in the query. In the real world, such specification is not easy, since users are not aware of what types are available. Our system allows type specification to be done based on keywords that match types, and all matching types are answer candidates; a match score for each answer type is taken into account along with entity scores, to get the overall answer ranking.

We use Lucene as a document-centric indexing system, and exploit our extended graph model to efficiently find entity mentions in proximity to keyword occurrences. The in-memory graph also provides a mapping between entities and their categories.

7. EXPERIMENTAL EVALUATION

In this section, we present a detailed analysis of the effectiveness our approach for solving near queries and entity-relationship queries. We look at the contribution of different factors involved in the approach. We also show a comparison of the quality of our results with those generated by the ERQ system of Li et al. [15].

7.1 Experimental Setup

We have implemented our algorithms in Java using servlets; we call our system WikiBANKS. Our system is available for access over the Web at the URL www.cse.iitb.ac.in/banks. The machine we used has 12 GB of RAM and an Intel Xeon

E5504, 2 GHz processor, with 1 TB Hard disk with SATA interface, running Ubuntu 10.04 LTS with a Linux 2.6.32-34 kernel. The database system used is PostgreSQL 8.3.7.

The Wikipedia graph was created out of a Wikipedia dump as of January 2009, and has following characteristics: number of nodes: 16.28 million, number of edges: 179.5 million, average indegree: 5.4303, maximum indegree: 374882. The graph takes about 4 GB space and takes about 85 seconds to load.

We index Wikipedia data using Lucene. For each document in Wikipedia, a virtual document is created, containing three fields: (a) Nodeid: the Wikipedia article ID, (b) Title: the title of the article, and (c) Content: the textual content of the article. The title and content field are indexed using Lucene, while Nodeid is stored but not indexed. The index stores term offsets for each occurrence of a term in each document that it occurs in. The index building is done after assigning prestige scores to the nodes of the graph. These node prestige scores are also included in indexing to boost the hits of the relevant nodes. We use Lucene *Collectors* to collect the Lucene scores for documents and *SpanQuery* to get the offsets of the search terms.

7.2 Query Set

Unlike the ERQ system of Li et al. [15], which supports only a limited number of categories, our system supports all the Yago categories, numbering nearly 150,000. Thus our system can answer a vastly larger number of queries than ERQ. However, to compare the two systems, we chose a set of 27 queries from the "Own28" set of Li et al. [15], available online at http://idir.uta.edu/erq/, as a performance benchmark. The query set includes:

- Q1 - Q16 : *Single selection predicate queries*, i.e. Near queries with only one selection predicate.
- Q17 - Q21 : *Multiple selection predicate queries*, i.e. Near queries with multiple selection predicates on the same entity variable.
- Q22 - Q27 : *Entity-relationship queries*, also known as multi-predicate queries with join.

For each query, we have a manually collected a set of correct answers, which we believe is fairly complete. We consider these sets as the ground truth when evaluating the performance. While [15] has only a limited number of entity categories available for use in queries, when we expressed the queries in our system for a few of the queries we made use of the richer set of categories available to us; the specific set of changes were: actor instead of person, football player instead of player, and football club instead of club, in queries where such substitutions are appropriate.

7.3 Parameter Settings

There are a number of input parameters involved in our query processing and scoring model. We have executed a large number of queries with different parameter settings, and manually chose the optimum values for these parameters, i.e. the values that gave the best precision. For selection predicates, we set the token span $\lambda = 12$, and the value $\sigma = 6$ for proximity scoring using the Gaussian kernel. For near queries with a single selection predicate we use the weightage for activation $\eta = 0.1$, while for near queries with multiple selection predicates we use $\eta = 0.6$. For entity-relationship queries, we set $\eta = 0.8$ for selection predicates; for relation predicates we set $\lambda = 16$, $\sigma = 8$, and the multiplicative weighting factor for relations predicates $\gamma = 0.6$.

7.4 Measures of Performance

We have used the following precision measures to compare the performance:

- **Precision at** k **:** Also referred to as $P@k$, it is the precision at a given cut-off rank. It is calculated as:

$$P@k = \frac{|relDocs \bigcap topKDocs|}{k}$$

  where *relDocs* is the set of all relevant documents (here, entities) and topKDocs is the set containing the top-K documents (here, entities) that are retrieved. Our precision at K graphs stop at $K = 10$.

- **Recall :** Recall is the fraction of the documents that are relevant to the query that are successfully retrieved:

$$recall = \frac{|relDocs \bigcap retrievedDocs|}{|relDocs|}$$

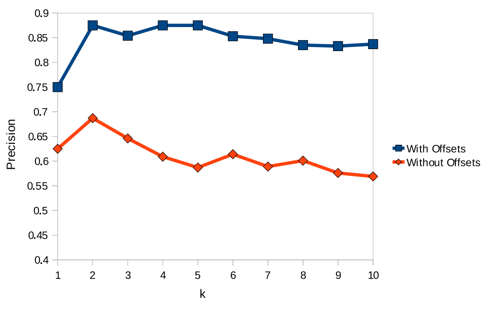
  where *retrievedDocs* is the set of all documents (here, entities) that are retrieved. To compare precision and recall, we have plotted precision at specific values of recall. To calculate this, we find the precision at the point when we have retrieved just enough answers to achieve a particular recall value (i.e. particular fraction of the set of all correct answers). When the system is unable to retrieve enough answers to achieve a particular recall value, we define the precision at that recall value as zero.

7.5 Experimental Results

Figure 1 compares the performance of the basic system (without the optimizations described in Section 5) with and without using offsets information. The comparison is for near queries Q1 through Q16. For the case where offsets are not used, we explore all nodes linked from the nodes containing near keywords, without regards to the token distance between the near keyword and the links. This causes a large number of irrelevant nodes to be explored, and increases query execution time as well as memory utilization. Figure 1 (a) shows that the average precision at k is much lower without using offset information, for k up to 10.

However, Figure 1 (b) indicates that the average precision is lower for "with offsets" than "without offsets" at 80% recall. This is because, in case of "without offsets", a large number of nodes are explored and hence it generates higher fraction of correct answers. The "with offsets" version spreads activation only to nodes that are within a limited offset (span), which is set to 12 by default. As a result this version explores fewer nodes, and fails to generate several answers which the "without offsets" technique is able to generate; as per our definitions, the precision is 0 at this point for these queries, reducing the average precision significantly. Since users are likely to only view the top-k results for some small value of k, the version with offsets is definitely preferable.

Next, we compare the performance of our system with ERQ [15]. We have experimented with 5 different versions of our system to isolate the effect of various optimizations described in Section 5. The different versions are as follows:


(a) Precision at k

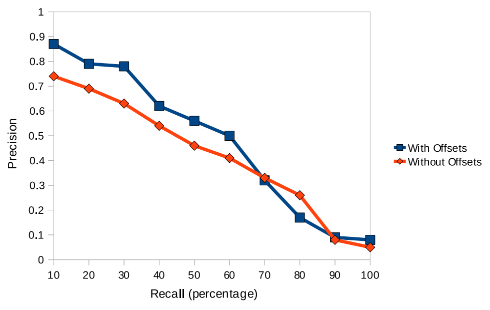

(b) Precision vs. recall

Figure 1: Effect of offsets on near queries with single selection predicate (no optimization)

k	Basic	Near Titles	Infobox	Near Categories	All 3	ERQ
1	0.704	0.666	0.814	0.851	0.851	0.741
2	0.741	0.777	0.759	0.833	0.814	0.833
3	0.703	0.728	0.753	0.79	0.814	0.796
4	0.731	0.75	0.741	0.796	0.833	0.75
5	0.733	0.748	0.733	0.807	0.822	0.76
6	0.703	0.715	0.703	0.802	0.814	0.716
7	0.693	0.714	0.692	0.793	0.804	0.72
8	0.675	0.694	0.689	0.777	0.81	0.734
9	0.678	0.691	0.695	0.765	0.802	0.71
10	0.681	0.685	0.696	0.751	0.785	0.698

Table 1: Precision at k for All Queries

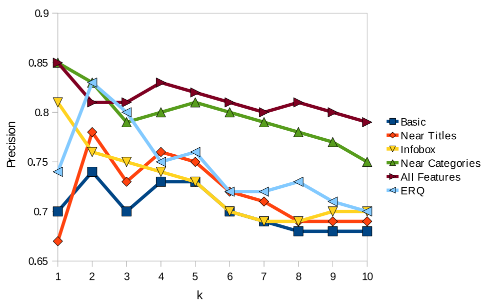


Figure 2: Precision at k for all queries

- **Basic:** In this version, we use the basic model without any of the optimizations.
- **Near Titles:** In this version, along with the basic features, we also spread activation from articles whose titles contain the near keywords to all its out-neighbors.
- **Infobox:** In this version, we use the infobox information and add some initial activation to the node whose infobox contains the near keywords.
- **Near Categories:** In this version, we exploit the Wikipedia category specificity as explained earlier.
- **All Features:** This version uses all the above optimizations along with the basic version.

The precision and precision versus recall numbers for ERQ are obtained by running the queries on their system, available online at http://idir.uta.edu/erq/.

Table 1 gives the precision at k values for our complete set of test queries. The table data clearly shows that each of the additional features improves the precision. Specially, the NearCategories feature improves the performance by a large margin. Using all the features together gives us the best performance.

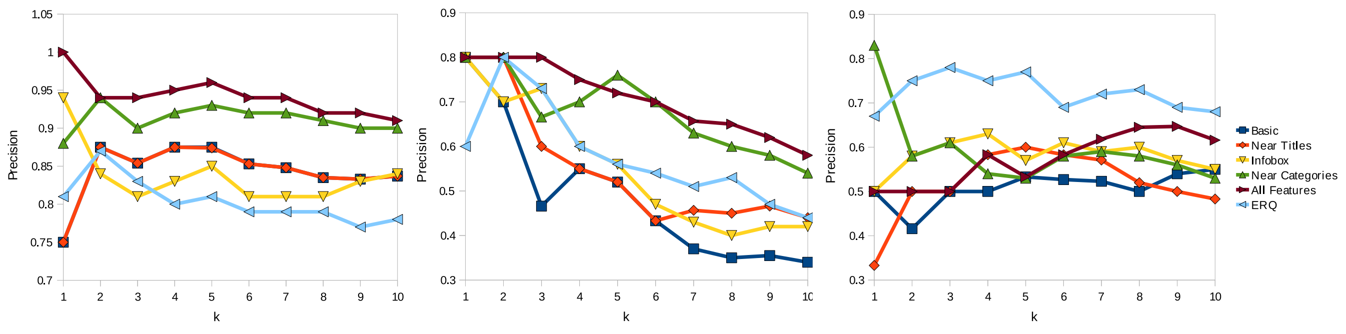
Figure 2 shows the plot for the precision at k values across all queries, with different system features turned on. Figure 3 shows the same information, but separately for different types of queries. The graphs indicates that our system clearly outperforms ERQ for near queries, with single selection predicate as well as with multiple selection predicates.

For entity-relationship queries, the ERQ system provided better precision. One reason for the lower precision is that our system allows flexible specification of categories. On Q28 from the OWN28 set (not included in the performance results) most of the films returned were in fact academy award winning movies adapted from novels, but in place of novels, the query returned other movies. This is because these movies are in categories such as "movies adapted from novels", and since such a category is treated as a valid category for the category keyword "novel", the query treats the movie itself as a novel. Requiring exact match for categories improved the result quality drastically, with 9 out the top 10 answers being correct. Our scoring system needs to be improved to avoid problems due to non-exact category matches.

However, the ERQ system requires categories to be precisely specified, which is not an easy task for a casual user, whereas we can handle queries where the categories are not precisely specified. (In addition our implementation can handle a very large number of categories in contrast to the limited number of categories handled by the current ERQ implementation.)

We also found anecdotally that the mutual exclusion and ordering pattern heuristics used in [15] would have been useful in improving precision, had we implemented them. Implementing these heuristics is an area of future work.

Table 2 shows average query execution time for various types of queries. Execution time is the response time measured when the query is input to the servlet. Table 3 shows average memory utilization for the queries in terms of num-



(a) near queries with single selection (b) near queries with multiple selections (c) entity-relationship queries

Figure 3: Precision at k by query type

Query Set	COLD CACHE	WARM CACHE
Single Selection	4.546	1.694
Multiple Selection	12.112	5.837
Entity-Relationship	14.44	9.317
All	8.233	4.284

Table 2: Average Query Execution Time (in seconds)

Query Set	Nodes Explored	Size of Target Queue
Single selection	7474	210
Multiple selection	48132	4074
Entity relationship	84003	9635
All	32010	3020

Table 3: Average Memory Utilization

Query Set	Basic	Near Titles	Info-box	Near Cate-gories	All 3	ERQ
Single selection	0.639	0.662	0.645	0.77	0.788	0.635
Multiple selection	0.448	0.488	0.448	0.565	0.598	0.414
Entity-relation-ship	0.511	0.537	0.500	0.511	0.533	0.672
All	0.575	0.602	0.577	0.674	0.696	0.602

Table 4: Average Recall

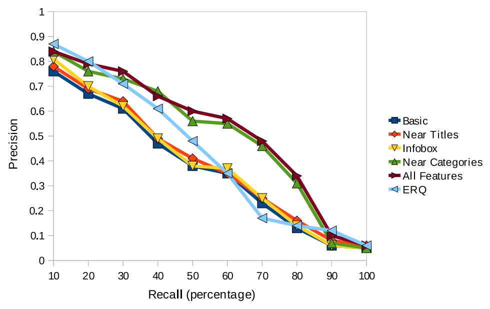


Figure 4: Precision vs. Recall for all queries

ber of nodes explored during activation spreading and size of the target queue. Target queue size determines the collection of nodes which are processed during ranking to produce final set of answers i.e. it is set of probable answers before ranking. Table 4 gives the average recall i.e. average of fraction of correct answers reported by the system for various types of queries.

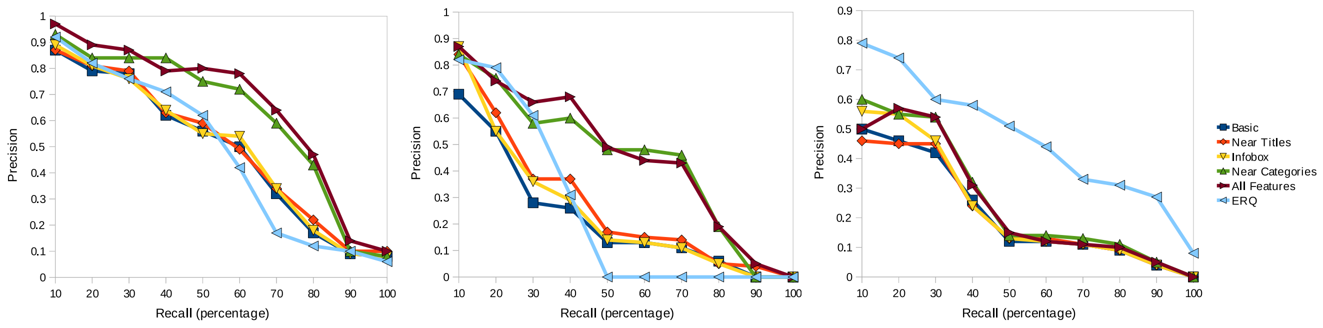
Figure 4 shows the plots of precision versus recall across all queries, while Figure 5 shows the same information for each query type. Since some queries do not have 100% recall up to the number of answers retrieved, we show the precision as 0 at recall percentages that are higher. Figure 4 show that WikiBANKS outperforms ERQ overall, while Figure 5 shows that WikiBANKS outperforms ERQ in case of near queries with single and multiple selection predicates, but ERQ achieves better performance for relationship queries.

We also tested our system on a number of other queries, many of which we could not run on the ERQ system since they used types such as medicines, airports, languages, animals, currencies, and so on which are among the many types not currently supported in the ERQ implementation. The precision at k and execution time for these queries were similar to the results we saw earlier for queries from the ERQ system. We omit details for lack of space.

We also ran some queries to test the impact of spreading activation through the graph, a feature we support, but other entity ranking techniques do not support. However, we did not find much difference since in most cases the Wikipedia corpus has direct links to the desired entities from pages containing the near keywords, and adding indirect activation did not help. For example, for the query "University near Nobel prize", the infoboxes of Nobel prize winners pages mentioned the Nobel prize and had a link to their institutions. We expect these results will be different if we include Web pages instead of just Wikipedia pages, an area of ongoing work.

8. CONCLUSIONS AND FUTURE WORK

We have proposed a novel extended graph representation,



(a) near queries with single selection (b) near queries with multiple selections (c) entity-relationship queries

Figure 5: Precision vs. recall by query type

and showed how to exploit it to answer entity ranking (near) queries and entity-relationship queries on the Wikipedia corpus. Unlike earlier systems we allow type specification through keywords, and develop novel scoring mechanisms based on spreading activation. Our performance study shows good result quality, beating earlier work on entity queries. Improving performance on entity-relationship queries is an area of current work.

The Wikipedia corpus has a limited number of explicit entity mentions, limiting the effect of spreading activation from keywords to nearby entities. We are currently extending our system to work on the annotated Web corpus of [6], which would provide a much richer set of keyword-entity associations. With such a system, we cannot keep the entire Web graph in memory; however, the number of entities is still relatively small (since we use Wikipedia as the source for entities), and we can keep these entities, along with their category hierarchy, in memory. We have developed versions of our algorithms tailored for such an environment, with data partitioned across multiple machines. Initial results demonstrate the feasibility of such a system, both in terms of answer quality, and interactive response times. We are currently working on improving the answer quality of the system. Extending our implementation to add the mutual exclusion and ordering pattern heuristics of [15] is another direction for future work.

Acknowledgment: This work was partially supported by IMPECS (Indo-German Max Planck Center for Computer Science).

9. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.

[2] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: authority-based keyword search in databases. In *VLDB*, 2004.

[3] H. Bast, A. Chitea, F. M. Suchanek, and I. Weber. Ester: efficient search on text, entities, and relations. In *SIGIR*, pages 671–678, 2007.

[4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.

[5] S. Chakrabarti, K. Puniyani, and S. Das. Optimizing scoring functions and indexes for proximity search in type-annotated corpora. In *WWW*, pages 717–726, 2006.

[6] S. Chakrabarti, D. Sane, and G. Ramakrishnan. Web-scale entity-relation search architecture. In *WWW (Companion Volume)*, pages 21–22, 2011.

[7] T. Cheng and K. C.-C. Chang. Beyond pages: Supporting efficient, scalable entity search with dual-inversion index. In *SIGMOD*, 2010.

[8] T. Cheng, X. Yan, and K. C.-C. Chang. EntityRank: Searching entities directly and holistically. In *VLDB*, 2007.

[9] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: Ranked keyword searches on graphs. In *SIGMOD*, pages 305–316, 2007.

[10] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *VLDB*, 2002.

[11] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.

[12] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: Searching and ranking knowledge. In *ICDE*, pages 953–962, 2008.

[13] S. Kulkarni, A. Singh, G. Ramakrishnan, and S. Chakrabarti. Collective annotation of wikipedia entities in web text. In *KDD*, pages 457–466, 2009.

[14] X. Li, C. Li, and C. Yu. Entityengine: answering Entity-Relationship queries using shallow semantics. In *CIKM*, pages 1925–1926, 2010.

[15] X. Li, C. Li, and C. Yu. Entity-relationship queries over wikipedia. *ACM Trans. on Intelligent Systems and Technology*, 3(4), Sept. 2012.

[16] Y. Lv and C. Zhai. Positional language models for information retrieval. In *SIGIR*, pages 299–306, 2009.

[17] J. Pound, I. F. Ilyas, and G. E. Weddell. Expressive and flexible access to web-extracted data: a keyword-based structured query language. In *SIGMOD Conf.*, pages 423–434, 2010.

[18] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago - a core of semantic knowledge. In *WWW*, 2007.

[19] M. A. Yosef, J. Hoffart, I. Bordino, M. Spaniol, and G. Weikum. AIDA: An online tool for accurate disambiguation of named entities in text and tables. *PVLDB*, 4(12):1450–1453, 2011.

Towards Efficient Discovery of Frequent Patterns with Relative Support

R. Uday Kiran and Masaru Kitsuregawa
Institute of Industrial Science,
University of Tokyo, Japan.
Email: uday_rage@tkl.iis.u-tokyo.ac.jp and kitsure@tkl.iis.u-tokyo.ac.jp.

ABSTRACT

Frequent patterns are an important class of regularities that exist in a database. Although there exists no universally acceptable best measure to assess the interestingness of a pattern, *relative support* is emerging as a popular measure to discover frequent patterns involving both frequent and rare items. An Apriori-like algorithm known as Relative Support Apriori (RSA) has been discussed in the literature to discover the patterns. It has been observed that mining frequent patterns with RSA is a computationally expensive process because the discovered patterns do not satisfy the anti-monotonic property. Moreover, RSA also suffers from the performance problems involving generation of the huge number of candidate patterns and multiple scans on the database. This paper makes an effort to discover frequent patterns effectively with the *relative support* measure. To reduce the computational cost, we theoretically show that the patterns discovered with the *relative support* measure satisfy the convertible anti-monotonic property. Using this property, a pattern-growth algorithm known as Relative Support Frequent Pattern-growth (RSFP-growth) has been proposed to discover the patterns. Experimental results on both synthetic and real-world datasets show that the proposed RSFP-growth algorithm is significantly better than the RSA algorithm.

1. INTRODUCTION

Frequent pattern mining is an important knowledge discovery technique in data mining. In the basic model of frequent patterns [3], a pattern (or an itemset) is considered frequent if it satisfies the user-defined minimum support (*minsup*) constraint. The *minsup* constraint controls the minimum number of transactions a pattern must cover in a database. Since only a single *minsup* constraint is used for the entire dataset, the model implicitly assumes that all items in a database have uniform frequencies. However, this is often not the case in many real-world databases. In many real-world applications, some items appear very frequently in the data, while others rarely appear. It has to be noted that considering an item in a database as either frequent or rare is a subjective issue which depends on the user and/or application requirements. If the items' frequencies in a database vary widely, we encounter the following issues:

i. If *minsup* is set too high, we will miss those patterns that involve rare items.

ii. In order to find the frequent patterns that involve both frequent and rare items, we have to set low *minsup*. However, this may cause combinatorial explosion, producing too many frequent patterns, because those frequent items will combine with one another in all possible ways and many of them can be meaningless depending upon the user and/or application requirements.

This dilemma is called the *rare item problem* [19]. One cannot ignore the knowledge pertaining to rare items. It is because such knowledge has been found useful in many real-world applications, such as detecting oil spills in satellite images [11] and improving the performance of recommender systems [5].

To confront the rare item problem in applications, numerous alternative measures have been discussed in the literature [4, 13, 18, 20, 21]. Unlike the *support* measure, these measures assess the interestingness of a pattern with respect to the frequencies of items within it. Each measure has a selection bias that justifies the significance of a knowledge pattern. As a result, there exists no universally acceptable best measure to judge the interestingness of a pattern for any given dataset or application. Researchers are making efforts to suggest a right measure depending upon the user and/or application requirements [17, 18, 20].

Yun et al. [21] have introduced the *relative support* measure and showed that it can discover the frequent patterns involving significant rare items effectively. A significant rare item represents an item that appears less frequently in the database (or having *support* less than the user-defined *minsup* threshold) but appears associated with the frequently occurring items in high proportion to its frequency. An Apriori-like algorithm known as Relative Support Apriori (RSA) algorithm has also been introduced to discover the patterns. We have observed that RSA is a computationally expensive algorithm because the patterns discovered with the *relative support* measure do not satisfy the anti-monotonic property. Moreover, RSA also suffers from the performance problems involving generation of the huge number of candidate patterns and multiple scans on the database.

This paper makes an effort to discover frequent patterns efficiently using the *relative support* and *support* measures. The contributions are as follows.

- The paper theoretically investigates the *relative support* measure and shows that the discovered patterns satisfy a property known as the *convertible anti-monotonic property* [15].

The 18th International Conference on Management of Data (COMAD), 14th-16th Dec, 2012 at Pune, India.

- Using the convertible anti-monotonic property and prior knowledge regarding the FP-growth construction and mining technique, we redefine the frequent pattern with a novel concept known as the *conditional minimum support*.
- Using the conditional minimum support, an FP-growth-like algorithm known as Relative Support Frequent Pattern-growth (RSFP-growth) has been proposed in this paper. Pei et al. [15] have theoretically shown that the search space for a pattern-growth algorithm discovering interesting patterns satisfying the convertible anti-monotonic property is same as the search space of a pattern-growth algorithm discovering interesting patterns satisfying the anti-monotonic property. Therefore, it can be said that the RSFP-growth algorithm reduces the computational cost of mining the patterns effectively.
- Experimental results on various datasets show that the RSFP-growth algorithm is runtime efficient and scalable than the RSA algorithm.

The rest of this paper is organized as follows. Section 2 discusses previous works on mining frequent patterns. Section 3 introduces the model of frequent patterns using the *relative support* and *support* measures. Section 4 discusses the performance issues of RSA algorithm, describes the basic idea and introduces the proposed RSFP-growth algorithm. Experimental evaluations of RSA and RSFP-growth algorithms are presented in Section 5. Finally, Section 6 concludes with future research directions.

2. RELATED WORK

Since the usage of single minimum support (*minsup*) framework to discover frequent patterns involving both frequent and rare items suffers from the dilemma known as the rare item problem, researchers have made efforts to discover frequent patterns with multiple *minsups* framework [7, 8, 9, 10, 12, 16], or other interestingness measures [4, 13, 18, 20, 21], or constraints [14].

In the multiple *minsups* framework, the user specifies a *minsup*-like constraint, called *minimum item support*, for every item in the database. Next, the *minsup* for a pattern is represented with the minimum item supports of the items within it. An open research problem of this framework is the methodology to determine the *minimum item supports* of all the items in a database.

Researchers have also introduced numerous alternative measures to discover frequent patterns. Examples includes *all-confidence*, *any-confidence*, *bond* [13], $lift$ and χ^2 [4]. Each measure has its own selection bias that justifies the significance of knowledge pattern. Thus, there exists no universally acceptable best measure to judge the interestingness of a pattern for any given application. In other words, selecting a right measure to discover frequent patterns involving both frequent and rare items is an important research issue. Researchers are making efforts to suggest a right algorithm depending upon the user and/or application requirements [17, 18, 20].

Yun et al. [21] have introduced the *relative support* measure to discover the frequent patterns involving significant rare items. An Apriori-like algorithm known as RSA has also been proposed to discover the complete set of patterns with the *relative support* and *support* measures [21]. The pattern model used in RSA requires three user-specified thresholds: first *support* (s_1), second *support* (s_2) such that $s_1 > s_2$ and minimum *relative support* (*minRsup*). The s_1 and s_2 thresholds are used to classify the items into either frequent or rare items. That is, items having *support* no less than s_1 are classified as **frequent items** and the items having *support* in between s_1 and s_2 are classified as **significant rare items**. If a pattern contains only frequent items, its *support* has to satisfy s_1 to be a frequent pattern. If an pattern contains rare items, its *support* must satisfy s_2 and its *relative support* must satisfy *minRsup* to be a frequent pattern. In many real-world applications, it is often difficult for the user to classify the items into either frequent or rare items. Therefore, a simplified approach is discover frequent itemsets using only s_2 and *minRsup* thresholds. In this paper, we use this approach to discover the frequent patterns. The performance issues of RSA algorithm and our efforts to address them are discussed in later parts of this paper.

3. THE FREQUENT PATTERN MODEL USING RELATIVE SUPPORT

The frequent pattern model using the *relative support* measure is as follows [21]:

Let $I = \{i_1, i_2, \cdots, i_n\}$ be a set of items, and DB be a database that consists of a set of transactions. Each transaction T contains a set of items such that $T \subseteq I$. Each transaction is associated with an identifier, called TID. Let $X \subseteq I$ be a set of items, referred as an itemset or a *pattern*. A pattern that contains k items is a k-pattern. A transaction T is said to contain X if and only if $X \subseteq T$. The frequency (or *support* count) of a pattern X in DB, denoted as $f(X)$, is the number of transactions in DB containing X. The *support* of X, denoted as $S(X)$, is the ratio of its frequency to the DB size, i.e., $S(X) = \frac{f(X)}{|DB|}$. The *relative support* of a pattern X, denoted as $Rsup(X)$, is the ratio of its *support* to the minimum *support* of an item (or 1-pattern) within it. That is, $Rsup(X) = \frac{S(X)}{min(S(i_j)|\forall i_j \in X)}$. The pattern X is **frequent** if its *support* and *relative support* are no less than the user-defined minimum *support* (*minsup*) and minimum *relative support* (*minRsup*) thresholds. That is, X is said to be frequent if

$$\begin{array}{rcl} S(X) & \geq & minsup \\ & and & \\ Rsup(X) & \geq & minRsup. \end{array} \quad (1)$$

We call the frequent pattern containing only one item as **frequent item**. We now illustrate the frequent pattern model using the transactional database shown in Table 1.

Table 1: Transactional database.

TID	Items	TID	Items
1	*a, b*	11	*a, b*
2	*a, b, e*	12	*a, c, f*
3	*c, d*	13	*a, b, e*
4	*e, f*	14	*b, e, f, g*
5	*c, d*	15	*c, d*
6	*a, c*	16	*a, b*
7	*a, b*	17	*c, d*
8	*e, f*	18	*a, c*
9	*c, d, g*	19	*a, b*
10	*a, b*	20	*c, d, f*

EXAMPLE 1. *The transactional database shown in Table 1 has 20 transactions. The set of items $I = \{a,b,c,d,e,f,g\}$. The set of items 'a' and 'b', i.e., $\{a, b\}$ is a pattern. It is a 2-pattern. For simplicity, we write this pattern as "ab". It occurs in 8 transactions (tids of $1,2,7,10,11,13,16$ and 19). Therefore, the support count of "ab," i.e., $f(ab) = 8$. The* support *of ab, i.e., $S(ab) = 0.4 \left(= \frac{8}{20}\right)$.*

The relative support *of ab, i.e.,* $Rsup(ab) = 0.88 \left(= \frac{0.4}{min(0.6, 0.45)}\right)$. *If the user-specified* $minsup = 0.3$ *and* $minRsup = 0.65$*, then ab is a frequent pattern because* $S(ab) \geq minsup$ *and* $Rsup(ab) \geq minRsup$.

It can be observed that the *relative support* measure assess the interestingness of a pattern with respect to the minimal *support* of an item within it. Thus, it can effectively discover all those frequent patterns that contain rare items occurring together with other items in high proportions of their frequencies.

The problem definition is as follows. Given the transactional database (*DB*), and the user-defined minimum *support* (*minsup*) and minimum *relative support* (*minRsup*) thresholds, discover the complete set of frequent patterns in a database that satisfy both *minsup* and *minRsup* thresholds.

4. PROPOSED APPROACH

In this section, we first discuss the performance problems of RSA algorithm. Subsequently, we introduce the basic idea and proposed RSFP-growth algorithm.

4.1 Performance problems of RSA algorithm

The *relative support* measure can effectively confront the rare item problem while mining frequent patterns involving both frequent and rare items. However, the RSA algorithm has the following issues.

- The RSA is a computationally expensive algorithm because the frequent patterns discovered with the *relative support* measure do not satisfy the *anti-monotic property*. That is, although a pattern satisfies the user-defined *minRsup* threshold, all its non-empty subsets may not have satisfied the *minRsup* threshold.
- It also suffers from the same performance problems as the Apriori algorithm, which includes generating huge number of candidate patterns and multiple scans on the database.

4.2 Basic Idea

The space of items in a database gives rise to a subset lattice. The itemset lattice is a conceptualization of the search space when mining frequent (or user-interest based) patterns. Reducing this search space is an important research issue in pattern mining. The popular techniques to reduce the search space involve the usage of anti-monotonic property, monotonic property, or succinct property (see Definition 1). If a constraint does not satisfy any of these properties, then it is said that mining patterns with it is a computationally expensive process as the mining algorithm has to search the entire lattice space, i.e., 2^I itemsets.

DEFINITION 1. *(Anti-monotone, Monotone and Succinct Constraints.) A constraint* C_a *is anti-monotone if and only if whenever a pattern S violates* C_a*, so does any superset of S. A constraint* C_m *is monotone if and only if whenever a pattern S satisfies* C_m*, so does any superset of S. Succinctness is defined in steps, as follows*

- *A pattern* $X \subseteq I$ *is a succinct set, if it can be expressed as* $\sigma_p(I)$ *for some selection predicate p, where* σ *is the selection operator.*
- $SP \subseteq 2^I$ *is a succinct powerset, if there is a fixed number of succinct sets* $I_1, I_2, \cdots, I_k \subseteq I$*, such that SP can be expressed in terms of the strict powersets of* $I_1, \cdots, I_k$ *using union and minus.*
- *Finally, a constraint* C_s *is succinct provided* $SAT_{C_s}(I)$ *is a succinct powerset.*

Pei et al. [15] have investigated the concept of convertible constraints, and showed that some of the constraints which do not satisfy any of these properties when items in a database are considered as an unordered set can satisfy these properties if items in a database are considered as an ordered set. These properties are known as the *convertible anti-monotonic* and *convertible monotonic* properties. All the constraints or measures discussed in the literature do not satisfy either of these two properties. Therefore, *identifying whether a constraint satisfies the convertible anti-monotonic property, convertible monotonic property, or neither of these two properties is a research problem in pattern mining.*

To reduce the computational cost, we have investigated the nature of *relative support* measure and found that it satisfies the convertible anti-monotonic property if items are arranged in *support* order. The correctness is based on Property 1 and Lemma 1 and is shown in Theorem 1. Unlike the anti-monotonic property, the definition of convertible anti-monotonic property varies with respect to the measure(s) and the order in which items are to be arranged to satisfy this property. Definition 2 defines the convertible anti-monotonic property for the patterns discovered using the *relative support* and *support* measures.

DEFINITION 2. *(**The convertible anti-monotonic property of a frequent pattern.**) If a* sorted k-pattern, $\{i_1, i_2, \cdots, i_k\}$, $k \geq 2$ *and* $S(i_1) \geq S(i_2) \geq \cdots \geq S(i_k)$*, is frequent, then all its non-empty subsets containing the item having lowest* support *within it (i.e.,* i_k*) will also be frequent. That is, if* $Rsup(X) \geq minRsup$ *and* $S(X) \geq minsup$*, then* $\forall Y \subseteq X$ *and* $i_k \in Y$*,* $Rsup(Y) \geq minRsup$ *and* $S(Y) \geq minsup$.

EXAMPLE 2. *In a transactional database, let xyz be a sorted frequent 3-pattern such that* $S(x) \geq S(y) \geq S(z)$*. Since* $xz \subset xyz$*, it turns out that xz is a frequent pattern as* $Rsup(xz) \geq Rsup(xyz) \geq minRsup$ *and* $S(xz) \geq S(xyz) \geq minsup$*. Similarly, yz is also a frequent pattern because* $Rsup(yz) \geq Rsup(xyz) \geq minRsup$ *and* $S(yz) \geq S(xyz) \geq minsup$*). The 1-pattern z is also a frequent pattern because* $S(z) \geq S(xyz) \geq minsup$*. Thus, in a sorted frequent 3-pattern xyz, all its non-empty subsets containing the item with lowest frequency within it (i.e., z, xz and yz) are also frequent. Please note that although* $xy \subset xyz$*, we cannot say* $Rsup(xyz)$ *will be less than or equal to* $Rsup(xy)$ *because there exists a case where* $\frac{S(xyz)}{S(z)} \not\leq \frac{S(xy)}{S(y)}$.

Pei et al. [15] have also theoretically shown that the convertible anti-monotonic property is same as the anti-monotonic property for a pattern-growth algorithm. Therefore, initially, we have extended the existing FP-growth [6] (or the *CFG* algorithm in [15]) to discover the complete set of frequent patterns using both *support* and relative *support* measures. It involved the following two steps:

i. Compress the database into the FP-tree, which retains the itemset association information.

ii. Using each interesting item in the FP-tree as an initial **suffix item** (or suffix pattern), construct its conditional pattern base consisting of the set of complete prefix paths in the FP-tree co-occurring with the suffix item, then construct its conditional FP-tree with all those items that have *support* and *relative support* no less than the respective *minsup* and *minRsup* thresholds, and perform mining recursively on such a FP-tree. The pattern-growth is achieved by the concatenation of the suffix pattern with the interesting patterns generated from the conditional FP-tree. The correctness of the algorithm is shown in Lemma 3.

We have observed that such approach is not an effective way to discover the complete set of frequent patterns as the every item in the conditional pattern base of a suffix pattern has to go through two checks, namely *minsup* and *minRsup*, to derive the corresponding conditional FP-tree. To reduce these number of checks, we redefine the frequent pattern using the notion of sorted set of items in a pattern. Definition 3 provides the alternative definition of a frequent pattern using the *support* and *relative support* measures. The correctness is shown in Lemma 2.

DEFINITION 3. *Given the user-specified minsup and minRsup constraints, a sorted k-pattern X,* $S(i_1) \geq S(i_2) \geq \cdots \geq S(i_k)$*, is said to be frequent if*

$$S(X) \quad \geq \quad max(S(i_k) \times minRsup,\ minsup). \tag{2}$$

EXAMPLE 3. *Continuing with Example 1, the pattern ab is frequent because*

$$\begin{aligned} S(ab) \quad &\geq \quad minsup \\ &and \\ \frac{S(ab)}{S(b)\ (= min(S(a), S(b)))} \quad &\geq \quad minRsup \end{aligned} \tag{3}$$

From Equation 3, it turns out that for the frequent pattern ab its $S(ab) \geq max(minRsup \times S(b),\ minsup)$.

Using Definition 3 and the prior knowledge regarding the FP-tree construction and mining technique, we introduce a novel concept known as the *conditional minimum support* (*Cminsup*). It is defined in Definition 4, and correctness is shown in Lemma 4.

DEFINITION 4. *(The conditional minimum support of a suffix pattern). Let* i_j *be the initial suffix item (or 1-pattern) having* support $S(i_j)$*. Let minsup and minRsup be the user-defined minimum* support *and minimum* relative support *thresholds. The conditional minimum* support *of a suffix pattern* $\alpha \ni i_j$*, denoted as* $Cminsup(\alpha)$*, is* $max(minRsup \times S(i_j),\ minsup)$.

Using the concept of conditional minimum *support*, we propose a pattern-growth algorithm known as Relative Support Frequent Pattern-growth (RSFP-growth), which is discussed in subsequent subsection.

PROPERTY 1. *(**Apriori property.**) If X and Y are the patterns such that* $Y \subset X$*, then* $S(Y) \geq S(X)$.

LEMMA 1. *Let* $X = \{i_1, i_2, \cdots, i_k\}$*,* $1 \leq k \leq n$*, be a pattern such that* $S(i_1) \geq S(i_2) \geq \cdots, S(i_k)$*. If* $Y \subset X$ *and* $i_k \in Y$*, then* $Rsup(Y) \geq Rsup(X)$.

PROOF. The *relative support* of X, i.e., $Rsup(X) = \frac{S(X)}{S(i_k)\ (=min(S(i_j)|\forall i_j \in X)}$. The *relative support* of Y, i.e., $Rsup(Y) = \frac{S(Y)}{S(i_k)\ (=min(S(i_j)|\forall i_j \in X)}$. Since $Y \subset X$, it turns out that $S(Y) \geq S(X)$ (Property 1). Thus, $Rsup(Y) \geq Rsup(X)$ as $\frac{S(Y)}{S(i_k)} \geq \frac{S(X)}{S(i_k)}$. □

THEOREM 1. *If the* relative support *of pattern satisfies the user-defined minRsup threshold, then all its non-subsets containing an item having the lowest* support *within it also satisfy the minRsup threshold.*

PROOF. Let $X = \{i_1, i_2, \cdots, i_k\}$, $1 \leq k \leq n$, be a pattern such that $S(i_1) \geq S(i_2) \geq \cdots \geq S(i_k)$. If $Rsup(X) \geq minRsup$, then

$$\frac{S(X)}{S(i_k)\ (= min(S(i_j)|\forall i_j \in X)} \quad \geq \quad minRsup. \tag{4}$$

If Y is a pattern such that $Y \subset X$ and $i_k \in Y$, then based on Lemma 1 it turns out that $Rsup(Y) \geq Rsup(X) \geq minRsup$. Therefore, if the *relative support* of a pattern satisfies the user-defined *minRsup* threshold, then all its subsets containing an item with lowest *support* within it will also satisfy the *minRsup* threshold. □

LEMMA 2. *Let* $X = \{i_1, i_2, \cdots, i_k\}$*,* $1 \leq k \leq n$*, be a pattern such that* $S(i_1) \geq S(i_2) \geq \cdots, S(i_k)$*. For the user-defined minsup and minRsup constraints, the pattern X can be said frequent if and only if* $S(X) \geq max(S(i_k) \times minRsup, minsup)$.

PROOF. From the definition of frequent pattern given in Equation 1, the pattern X can be said frequent if

$$\begin{aligned} S(X) \quad &\geq \quad minsup \\ &and \\ \frac{S(X)}{min(S(i_1), S(i_2), \cdots, S(i_k))} \quad &\geq \quad minRsup. \end{aligned} \tag{5}$$

Since $min(S(i_1), S(i_2), \cdots, S(i_k)) = S(i_k)$, Equation 5 can be expressed as follows:

$$\begin{aligned} S(X) \quad &\geq \quad minsup \\ &and \\ \frac{S(X)}{S(i_k)} \quad &\geq \quad minRsup. \end{aligned} \tag{6}$$

Thus, X can be a frequent patten if and only if $S(X) \geq max(S(i_k) \times minRsup, minsup)$. □

LEMMA 3. *Let* α *be a suffix pattern in* FP-tree*. Let* $min\_item\_sup(\alpha)$ *be the minimum* support *of an item in* α*, i.e.,* $min\_item\_sup(\alpha) = min(S(i_j)|\forall i_j \in \alpha)$*. Let B be* α *conditional pattern base, and* β *be an item in B. Let* $S(\beta)$ *be the* support *of* β *in the transactional database. Let* $S_B(\beta)$ *be the **conditional support** of* β*, i.e.,* support *of* β *in B, respectively. If* α *is frequent,* $S_B(\beta) \geq minsup$ *and* $\frac{S_B(\beta)}{min\_item\_sup(\alpha)} \geq minRsup$*, then the pattern* $<\alpha, \beta>$ *is also a frequent pattern.*

PROOF. According to the definition of conditional pattern base and FP-tree, each subset in B occurs under the condition of the occurrence of α in the transactional database. If an item β appears in B for n times, it appearers with α in n times. Further, $min\_item\_sup(\alpha) = min\_item\_sup(\alpha \cup \beta)$ as FP-tree is constructed in *support* descending order of items. Thus, from the definition of frequent pattern used in this model, if the $S_B(\beta) \geq minsup$ and $\frac{S_B(\beta)}{min\_item\_sup(\alpha)} \geq minRsup$, the pattern $<\alpha, \beta>$ is therefore a frequent pattern. □

LEMMA 4. *Let* α *be a suffix pattern in* FP-tree *that has resulted from the initial suffix item* i_j*. Let the* $Cminsup(\alpha)$ *be the Cminsup of* α*. Let B be* α *conditional pattern base, and* β *be an item in B. Let* $S(\beta)$ *and* $S_B(\beta)$ *be the* support *of* β *in the transactional database and in B, respectively. If* α *is frequent and* $S_B(\beta) \geq Cminsup$*, the pattern* $<\alpha, \beta>$ *is therefore also frequent.*

PROOF. From the mining procedure of FP-tree, the $min\_item\_sup(\alpha) = S(i_j)$. From Lemma 3, it turns out that if the

$S_B(\beta) \geq minsup$ and $\frac{S_B(\beta)}{S(i_j)} \geq minRsup$, the pattern $< \alpha, \beta >$ is a frequent pattern. In other words, $< \alpha, \beta >$ is a frequent pattern if $S_B(\beta) \geq max(minsup, \text{-}\ minRsup \times S(i_j))$. From the definition of conditional minimum *support*, it can be said that $< \alpha, \beta >$ is a frequent if $S_B(\beta) \geq Cminsup(\alpha)$ $(= max(minsup,\ minRsup \times S(i_j)))$. $\square$

4.3 Relative Support Frequent Pattern-growth

The RSFP-growth algorithm uses pattern-growth technique and the concept of conditional minimum *support* to discover frequent patterns effectively using the *relative support* and *support* measures. Briefly, the RSFP-growth involves the following two steps:

i. Compress the database into the FP-tree, which retains the itemset association information.

ii. Using each interesting item in the FP-tree as an initial **suffix item** (or suffix pattern), measure the conditional minimum *support* (*Cminsup*) and construct its conditional pattern base consisting of the set of complete prefix paths in the FP-tree co-occurring with the suffix item. Next, construct the conditional FP-tree with all those items that have *support* no less than the *Cminsup* threshold in the conditional pattern base, and perform mining recursively on such a FP-tree using *Cminsup*. The pattern-growth is achieved by the concatenation of the suffix pattern with the interesting patterns generated from the conditional FP-tree.

The working of RSFP-growth is shown in Algorithm 1 and described as follows. The RSFP-growth algorithm accepts transactional database, *minsup* and *minRsup* as its input parameters. An FP-tree [6] is created using *minsup* threshold (line 1 in Algorithm 1). Next, the procedure RSFP-mine_1 is called to discovered frequent patterns from FP-tree. The RSFP-mine_1 procedure selects each item in the FP-tree as an initial suffix item (or pattern) and calculates its *Cminsup* (line 2 in Procedure 2). Next, the conditional pattern base and conditional FP-tree are generated for the suffix item using *Cminsup* (lines 3 to 11 in Procedure 2)). Next, the RSFP-mine_k procedure is called to recursively mine frequent patterns from the conditional FP-tree of suffix item.

We now explain the working of RSFP-growth algorithm using the database shown in Table 1. Let the user-defined *minsup* and *minRsup* thresholds be 3 and 0.65, respectively. Scan the database and measure the *support* of items in a database. Prune the infrequent items (i.e., items having *support* less than *minsup*) and sort the remaining items in the order of descending *support*. This resulting set or *list* is denoted *L*. Thus, we have $L = \{\{a:11\}, \{b:9\}, \{c:9\}, \{d:6\}, \{e:5\}, \{f:5\}\}$.

An FP-tree is constructed as follows. First, create the root of the tree, labeled with "null." Scan database *DB* a second time. The scan of the first transaction, "1:a, b," which contains two items (a and b in L order), leads to the construction of the first branch of the tree with two nodes, $\langle a:1 \rangle$ and $\langle b:1 \rangle$, where a is linked as a child of the root and b is linked to a. The second transaction, "2:a, b, e," would result in a branch where a is linked to root, b is linked to a and e is linked to b. However, this branch would share a common prefix, a and b, with the existing path for "1". Therefore, we instead increment the count of a and b by 1, and create a new node, $\langle e:1 \rangle$, which is linked as the child of $\langle b:2 \rangle$. Similar process is repeated for other transactions in the database. To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links. The FP-tree obtained after scanning all transactions of Table 1 is shown in Figure 1 with the associated node-links.

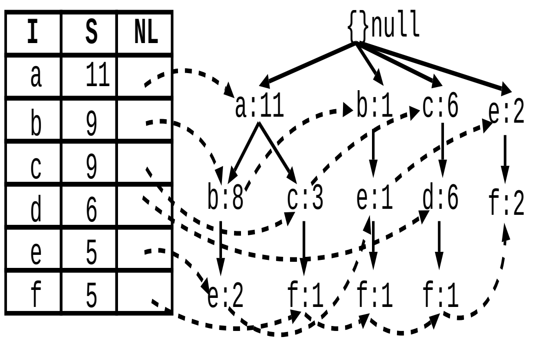


Figure 1: FP-tree. The terms 'I', 'S' and 'NL' respectively denote item, support and node-link.

Table 2: Mining frequent patterns.

SI	*Cmin-sup*	Conditional Pattern Base	Conditional FP-tree	Frequent patterns
f	3	$\{\{a,c:1\}, \{c,d:1\}, \{e:2\}, \{b,e:1\}\}$	$\langle e:3 \rangle$	{e, f : 3}
e	3	$\{a,b:2\}$	-	-
d	3	$\{c:6\}$	$\langle c:6 \rangle$	{c, d : 6}
c	3	$\{a:3\}$	-	-
b	3	$\{ea:8\}$	$\langle a:8 \rangle$	{a, b : 8}

Mining frequent patterns using FP-tree of Figure 1 is shown in Table 2 and detailed as follows. Consider the item f, which is the last item in L, as a suffix item. The item f occurs in four branches of the FP-tree of Figure 1. The *Cminsup* of 'f' is 3 ($\simeq max(3, 5 \times 0.65)$. The paths containing f in FP-tree are $\langle a,c,f:1 \rangle$, $\langle c,d,f:1 \rangle$ $\langle e,f:2 \rangle$ and $\langle b,e,f:1 \rangle$. Therefore, considering f as a suffix, its corresponding four prefix paths are $\langle a,c:1 \rangle$, $\langle c,d:1 \rangle$, $\langle e:2 \rangle$ and $\langle b,e:1 \rangle$, which form its conditional pattern base. Its conditional FP-tree contains only a single path, $\langle e:3 \rangle$. The items a, b, c and d are not included in conditional FP-tree because their *support* of 1 is less than *Cminsup*. The concatenation of suffix pattern with the item in conditional FP-tree generates the frequent pattern $\{e, f:3\}$. Similar process is repeated for the remaining other items in the FP-tree to discover the complete set of frequent patterns.

5. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of FP-growth, RSA and RSFP-growth algorithms. We show that RSFP-growth is a better algorithm to mine frequent patterns in different types of datasets.

The algorithms are written in GNU C++ and run with the Ubuntu 10.04 operating system on a 2.66 GHz machine with 1GB memory. The runtime specifies the total execution time, i.e., CPU and I/Os. The runtime is expressed in seconds. We pursued experiments on synthetic (T10I4D100K) and real-world (BMS-WebView-1, Mushroom and Kosarak) datasets. The datasets are available at Frequent Itemset Mining repository [1]. The details of these datasets are shown in Table 3.

5.1 Generation of Frequent Patterns

Figure 2(a), (b) and (c) respectively show the number of frequent patterns generated in $T10I4D100k$, *BMS-WebView-1* and *Mushroom* datasets with the basic model (denoted as *BM*) [2] and the

Table 3: Dataset Characteristics. The terms "Max.", "Avg." and "Tran." are respectively used as the acronyms for "maximum", "average" and "transaction."

Dataset	Transa-ctions	Distinct Items	Max. Trans. Size	Avg. Trans. Size	Type
*T*10*I*4*D*100*k*	100000	870	29	10.1	sparse
BMS-WebView-1	59602	4971	267	2.5	sparse
Mushroom	8124	119	23	23.0	dense
Kosarak	990002	41270	2498	8.1	sparse

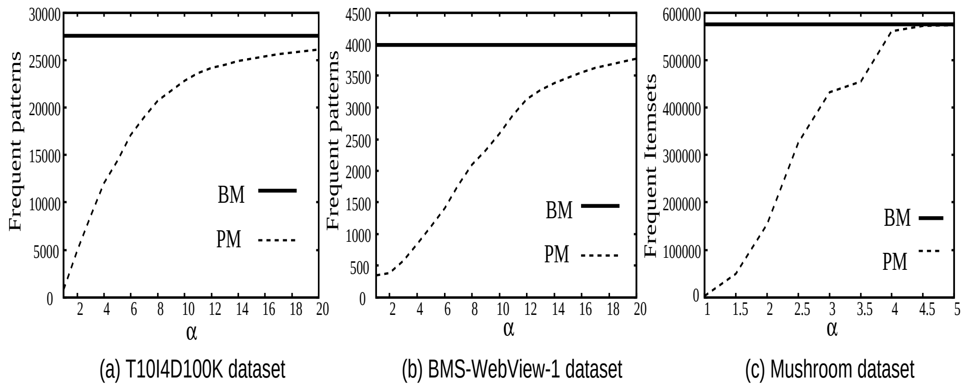


Figure 2: Frequent patterns generated in different databases at different *minRsup* values.

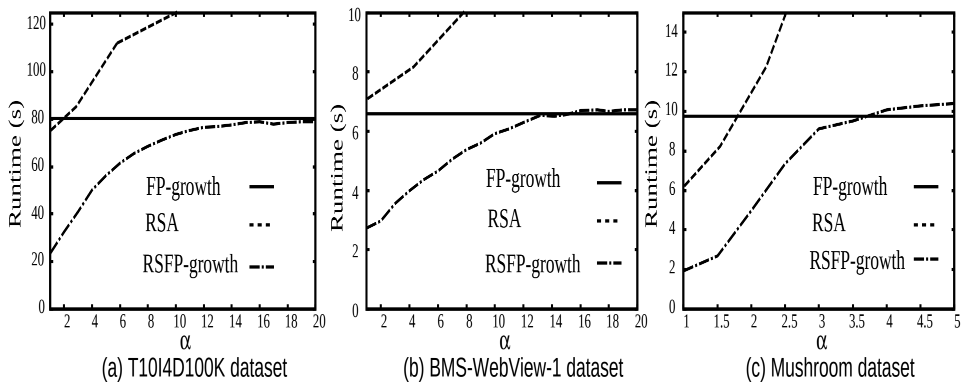


Figure 3: Runtime consumed by different algorithms in different databases at different *minRsup* values.

Algorithm 1 RSFP-growthAlgorithm (*DB*: database, *minsup*: minimum support, *minRsup*: minimum relative support)

```
1: The FP-tree is constructed in the following steps:
     i. Scan the transactional database $D$ once. Collect $F$, the set of frequent items, and their support counts. Sort $F$ in support descending order as $L$, the list of frequent items.
     ii. Create the root of an FP-tree, and label it as "null." For each transaction $t$ in $D$ do the following. Select and sort the frequent items in $t$ according to the order of $L$. Let the sorted frequent item list in $t$ be $[p|P]$, where $p$ is the first element and $P$ is the remaining list. Call insert_tree($[p|P], T$), which is performed as follows. If $T$ has a child $N$ such that $N.item\text{-}name = p.item\text{-}name$, then increment $N$'s count by 1; else create a new node $N$, and let its count be 1, its parent link be linked to $T$, and its node-link to the nodes with the same item-name via the node-link structure. If $P$ is non-empty, call insert_tree($P, N$) recursively.
2: The FP-tree is mined by calling RSFP-mine_1($Tree, null$).
```

proposed model (denoted as *PM*) of frequent patterns. The *minsup* thresholds set in $T10I4D100k$, *BMS-WebView-1* and *Mushroom* are 0.1%, 0.1% and 25%, respectively. The *minRsup* threshold in each database is set as $\frac{1}{\alpha}$ and varied α from 1 to 20. The thick lines shows the number of frequent patterns discovered by FP-growth (or basic model). It can be observed that increase in α value has increased the number of frequent patterns in the proposed model. The reason is due to the decrease in *minRsup* threshold with the increase in α value. If $minRsup = 0$ (or α is set too large), then both the models will generate same number of frequent patterns.

5.2 Runtime Comparison of FP-growth, RSA and RSFP-growth Algorithms

Figure 3(a), (b) and (c) respectively show the runtime taken by FP-growth, RSA and FP-growth algorithms in $T10I4D100k$, *BMS-WebView-1* and *Mushroom* datasets. The *minsup* thresholds set in $T10I4D100k$, *BMS-WebView-1* and *Mushroom* are 0.1%, 0.1% and 25%, respectively. The *minRsup* threshold in each database is set as $\frac{1}{\alpha}$ and varied α from 1 to 20. To compare RSA and RFP-growth algorithms, the s_1 and s_2 thresholds of *RSA* algorithm are respectively set to 100% and *minsup* of the database. The following observations can be drawn from these figures.

- Since the number of frequent patterns getting generated with FP-growth remained constant, the runtime of FP-growth has resulted in a straight line.
- Increase in α value has increased the runtime of both RSA and RSFP-growth algorithms. The reason is due to the increase of number of frequent patterns with increase in α value.
- The RSFP-growth algorithm has taken relatively less runtime than the FP-growth algorithm at lower α (i.e., higher *minRsup* values). It is because of the less number of frequent patterns discovered by RSFP-growth.
- At higher α (i.e., at low *minRsup*), the FP-growth and RSFP-growth algorithms have discovered almost same the number of frequent patterns. However, the runtime of RSFP-growth is slightly more than the FP-growth. It is because of the additional runtime was spent by RSFP-growth algorithm in calculating the *conditional minimum support* for each suffix item.

  Please note that the runtime of RSFP-growth can be much higher than FP-growth if *conditional minimum support* is not used.
- At any α value (i.e., irrespective of *minRsup* threshold), RSFP-growth is better than the RSA algorithm. It is because of two reasons: first, the search space of RSA was more than the search space of RSFP-growth algorithm; second, RSA suffered from the same performance problems as the Apriori algorithm.

Procedure 2 RSFP-mine_1($Tree, \alpha$); Constructing the conditional pattern base for frequent item or length-1 suffix pattern.

```
1: for each $a_i$ in the header of $Tree$ do
2:     Calculate $Cminsup = max(minsup, S(a_i) \times minRsup)$.
3:     Generate pattern $\beta = \alpha \cup a_i$ with $support = a_i.support$. {The term support represent support count.} {$S(\beta) = S(a_i)$ in $\alpha$-projected database}
4:     Get a set $I_\beta$ of items to be included in $\beta$-projected database.
5:     for each item in $I_\beta$, compute its support in $\beta$-projected database;
6:     for each $b_j \in I_\beta$ do
7:         if $S(\beta b_j) < Cminsup$ then
8:             delete $b_j$ from $I_\beta$; {pruning based on conditional minimum support}
9:         end if
10:    end for
11:    construct $\beta$-conditional FP-tree with items in $I_\beta$ $Tree_\beta$.
12:    if $Tree_\beta \neq \emptyset$ then
13:        RSFP-mine_k($Tree_\beta, Cminsup, \beta$);
14:    end if
15: end for
```

Procedure 3 RSFP-mine_k($Tree, Cminsup, \alpha$); Constructing the conditional pattern base for length-k, $k > 1$, suffix pattern.

```
1: for each $a_i$ in the header of $Tree$ do
2:     Generate pattern $\beta = \alpha \cup a_i$ with $support = a_i.support$.
3:     Get a set $I_\beta$ of items to be included in $\beta$-projected database.
4:     for each item in $I_\beta$, compute its support in $\beta$-projected database;
5:     for each $b_j \in I_\beta$ do
6:         if $S(\beta b_j) < Cminsup$ then
7:             delete $b_j$ from $I_\beta$; {pruning based on minimum support}
8:         end if
9:     end for
10:    construct $\beta$-conditional FP-tree with items in $I_\beta$ $Tree_\beta$.
11:    if $Tree_\beta \neq \emptyset$ then
12:        RSFP-mine_k($Tree_\beta, Cminsup, \beta$);
13:    end if
14: end for
```

5.3 Scalability Test on RSA and RSFP-growth Algorithms

In this experiment, we evaluate the scalability performance of RSA and RSFP-growth algorithms on runtime requirements by varying the number of transactions in a database. We use real-world *kosarak* dataset for the scalability experiment, since it is a huge sparse dataset. We divided the dataset into five portions of 0.2 million transactions in each part. Then we investigated the performance of RSA and RSFP-growth algorithms after accumulating

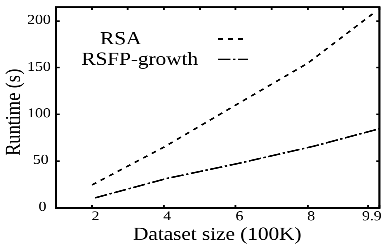


Figure 4: Scalability of RSA and RSFP-growth algorithms.

each portion with previous parts while performing correlated pattern mining each time. We fixed *minsup* $= 0.1\%$ and *minRsup* $=$ 0.5 (i.e., $\alpha = 2$) for each experiment. The experimental results are shown in Figure 4. The runtime in y-axes of Figure 4 specify the total runtime consumed by RSA and RSFP-growth algorithms with the increase of database size. It is clear from the graphs that as the database size increases, overall runtime increases for both RSA and RSFP-growth algorithms. However, the RSFP-growth algorithm requires relatively less runtime than RSA algorithm. Therefore, it can be observed from the scalability test that RSFP-growth can efficiently mine frequent patterns over large datasets and distinct items with considerable amount of runtime.

Overall, the RSFP-growth algorithm is runtime efficient and scalable than the RSA algorithm.

6. CONCLUSIONS AND FUTURE WORK

This paper has proposed an efficient and effective pattern-growth algorithm to discover the complete set of frequent patterns using *relative support* and *support* measures. The paper has also shown that it is not computationally expensive to mine the patterns as the *relative support* measure satisfies the *convertible anti-monotonic property* if items within a pattern are arranged in *support* order. A novel concept known as conditional minimum *support* has been introduced and extended to FP-growth algorithm to discover frequent patterns. By conducting experiments on various datasets, we have shown that RSFP-growth outperforms RSA algorithm with respect to both runtime and scalability.

The future works of the paper are as follows: first, data mining techniques, such as classification and clustering, employ frequent patterns discovered with single *minsup* constraint to improve their performance. As a result, these techniques also suffer from the rare item problem. It is interesting to investigate the usage of frequent patterns discovered with the relative support measure to address the problem. Second, the interestingness of frequent patterns discovered using various measures, such as *relative support* and *all-confidence*, needs to be investigated.

7. REFERENCES

[1] Frequent itemset mining repository. http://fimi.cs.helsinki.fi/data/ .

[2] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, New York, NY, USA, 1993. ACM.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, 1994.

[4] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: generalizing association rules to correlations. *SIGMOD Rec.*, 26(2):265–276, 1997.

[5] F. Gedikli and D. Jannach. Neighborhood-restricted mining and weighted application of association rules for recommenders. In *Proceedings of the 11th international conference on Web information systems engineering*, WISE'10, pages 157–165, Berlin, Heidelberg, 2010. Springer-Verlag.

[6] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, 2004.

[7] R. U. Kiran and P. K. Reddy. An improved frequent pattern-growth approach to discover rare association rules. In *KDIR*, pages 43–52, 2009.

[8] R. U. Kiran and P. K. Reddy. Improved approaches to mine rare association rules in transactional databases. In *IDAR '10: Proceedings of the Fourth SIGMOD PhD Workshop on Innovative Database Research*, pages 19–24, New York, NY, USA, 2010. ACM.

[9] R. U. Kiran and P. K. Reddy. Mining rare association rules in the datasets with widely varying items' frequencies. In *DASFAA (1)*, pages 49–62, 2010.

[10] R. U. Kiran and P. K. Reddy. Novel techniques to reduce search space in multiple minimum supports-based frequent pattern mining algorithms. In *EDBT*, pages 11–20, 2011.

[11] M. Kubat, R. C. Holte, and S. Matwin. Machine learning for the detection of oil spills in satellite radar images. *Mach. Learn.*, 30(2-3):195–215, Feb. 1998.

[12] B. Liu, W. Hsu, and Y. Ma. Mining association rules with multiple minimum supports. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 337–341, New York, NY, USA, 1999. ACM.

[13] E. R. Omiecinski. Alternative interest measures for mining associations in databases. *IEEE Trans. on Knowl. and Data Eng.*, 15(1):57–69, 2003.

[14] J. Pei and J. Han. Constrained frequent pattern mining: a pattern-growth view. *SIGKDD Explor. Newsl.*, 4(1):31–39, 2002.

[15] J. Pei, J. Han, and L. V. Lakshmanan. Pushing convertible constraints in frequent itemset mining. *Data Mining and Knowledge Discovery*, 8:227–252, 2004. 10.1023/B:DAMI.0000023674.74932.4c.

[16] C. S. K. Selvi and A. Tamilarasi. Mining association rules with dynamic and collective support thresholds. *International Journal on Open Problems Computational Mathematics*, 2(3):427–438, 2009.

[17] A. Surana, R. U. Kiran, and P. K. Reddy. Selecting a right interestingness measure for rare association rules. In *COMAD*, page 115, 2010.

[18] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 32–41, New York, NY, USA, 2002. ACM.

[19] G. M. Weiss. Mining with rarity: a unifying framework. *SIGKDD Explor. Newsl.*, 6(1):7–19, 2004.

[20] T. Wu, Y. Chen, and J. Han. Re-examination of interestingness measures in pattern mining: a unified framework. *Data Min. Knowl. Discov.*, 21(3):371–397, 2010.

[21] H. Yun, D. Ha, B. Hwang, and K. H. Ryu. Mining association rules on significant rare data using relative support. *J. Syst. Softw.*, 67:181–191, September 2003.

D E M O N S T R A T I O N

Excel Solvers for the Traveling Salesman Problem

Mangesh Gharote, Dilys Thomas, Sachin Lodha
mangesh.g@tcs.com dilys@cs.stanford.edu sachin.lodha@tcs.com
Tata Consultancy Services, Pune, India

ABSTRACT

Ordering queries within a workload and ordering joins in a query are important problems in databases [1]. We give algorithms for the query sequencing problem that scale (small space) and are efficient (low runtime) as compared to earlier work [4]. The errors are small in practice and we are able to further reduce them using geometric repair. We provide a computational comparison of TSP solvers and show extensive testing on benchmark datasets [25] observing its connection to these ordering problems.

1. PROBLEM STATEMENT

Database systems are facing an ever increasing demand for high performance. Either as standalone Oracle, SQLServer or DB2 installations or as a backend to Peoplesoft, SAP or Siebel workloads they are required to execute a batch of queries that contain several common subexpressions. Traditionally, query optimizers like [37], [36] optimize queries one at a time and do not identify any commonalities in queries, resulting in repeated computations. As observed in [3, 39] exploiting common results, multi-query optimization (MQO), can lead to significant performance gains – this requires the queries to be ordered in the workload for memory reuse and reduced disk need. Motivated by the importance for ordering problems, we study the combinatorial ordering problem of the travelling salesman problem (TSP) and provide extensive testing on benchmark datasets [25].

1.1 Applications

The traveling salesman problem has wide applicability in many different industrial and scientific scenarios. Some notable ones are: vehicle routing, bus scheduling, development of flight schedules, crew scheduling, order-picking problem in warehouses, printing press scheduling problem, network cabling in a country, computer wiring, query workload ordering for optimization, VLSI chip design connectivity layout, drilling of printed circuit boards, genome sequencing, hot rolling scheduling problem in iron & steel industry, overhauling gas turbine engines , X-Ray crystallography (ordering positions for measurement), global navigation satellite system, ordering test cases in regression suite to re-use components etc.

The 18th International Conference on Management of Data (COMAD), 14th-16th Dec, 2012 at Pune, India.

See [6] for a description of some applications of TSP. Intractability [12] [11] and restricted tractability results [9] [10] for TSP have won top awards. We develop our own algorithms on top of reasonable in-practice TSP algorithms. We obtain near optimal tours in practice. Our aim is to reduce run time and be scalable in memory for medium to large instances of TSP. Ease of using the tool, ability to handle different distance metrics including longitude and latitude, and ease of visualizing the tours produced are the aims of our project of improving state of the art TSP solvers available in Excel [4].

2. NEAREST NEIGHBOR AND GREEDY ALGORITHMS

2.1 Nearest Neighbor

Algorithm 1 implemented in our Excel solver is the Nearest Neighbor(NN) algorithm. Since it grows a single segment, it is similar to left deep plans used in query optimizers. Different start points can give different tours, see Figure 1.

Algorithm 1 Nearest Neighbor
```
Select an arbitrary vertex as current vertex.
while not all the vertices in domain are visited do
    Find shortest edge connecting current vertex and an unvisited
    vertex V.
    Set current vertex to V. Mark V visited.
end while
```

2.2 Greedy

Instead of starting from one vertex in NN, Algorithm 2 the greedy algorithm grows multiple segments and stitches them together to get a tour, similar to bushy optimizer plans.

Algorithm 2 Greedy
```
Sort all edges.
while less than n edges in tour do
    Select the shortest edge and add it to tour if
    [1] not yet on tour and not creating a degree-3 vertex.
    [2] not creating a cycle of size less than n.
end while
```

3. TOUR REPAIR

NN cannot approximate TSP to better than a factor of $log(n)$ [40] and may produce the worst possible tour [13]. In practice NN and

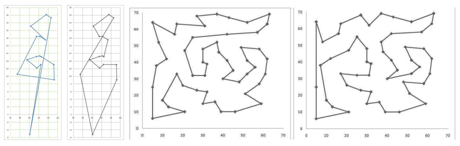


Figure 1: Different start points in 16 NN(32% from opt),(5% from opt), 51 NN, Greedy(intersection removal, section 3.1)(8% from opt),(11% from opt)

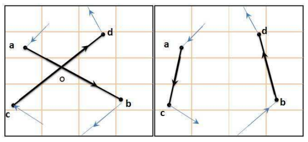


Figure 2: Intersection Unrolling

greedy gives within 25% away from optimal for moderately large sized instances. See Figures 6, 7, 8. The solutions obtained can be further repaired with our intersection removal, hinge-crest optimization, and tested techniques like geometric constructions, k-opt, etc.

3.1 Intersection Unrolling

From Figure 2 (i) Triangle Inequality $ao + co > ac$. (ii) Again $do + ob > db$. (iii) Adding (i) & (ii) $ao + co + do + bo > ac + db$. (iv) Rearranging terms $ao + ob + co + do > ac + db$. (v) Intersection Unrolling $ab + cd > ac + db$. We solve for the intersection point using Cramers Rule. Intersection unrolling is applied when intersection point lies on both segments, as shown in Algorithm 3. For every i, j intersection, the tour between vertices Tour[i+1] and Tour[j] has been reversed by the inner while. See Figure 3 for examples.

Algorithm 3 Unroll Intersection

```
while (Tour[i],Tour[i+1]) (Tour[j],Tour[j+1]) intersect do
    L = i + 1. R = j.
    while L < R do
        Swap = Tour[L]. Tour[L] = Tour[R]. Tour[R] = Swap.
        L = L + 1. R = R - 1.
    end while
end while
```

3.2 Hinge and Crest Optimization

The hinge and crest optimization (transfer tour repair) from Figure 4 is given in Algorithm 4 and applied in Figure 5.

4. RELATED WORK AND EXPERIMENTS

Being the most important geometric combinatorial problem, the TSP has multiple popular algorithms.

4.1 Lin-Kernighan

Lin-Kernighan heuristic tries removing k edges and adding k other edges aiming to retain a tour but to reduce the cost taking at most $O(n^k)$ time.

4.2 Linear Programming Formulation, Cutting Plane

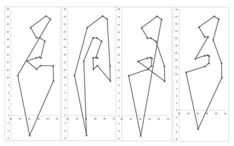


Figure 3: Intersection Removal on 16 NN(5% from opt),(3% from opt) and 16 Greedy(17% from opt),(1% from opt)

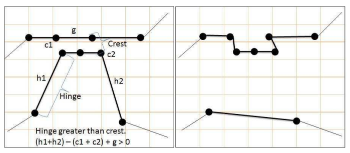


Figure 4: Hinge and Crest Transfer

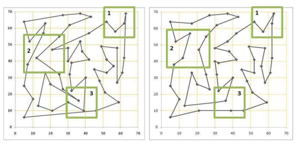


Figure 5: Hinge and Crest Transfer, 51 points

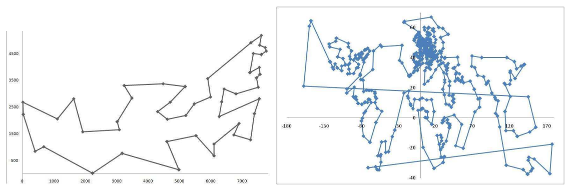


Figure 6: 48 US mainland capitals(our 7%),6 continents 535 airports(10%)

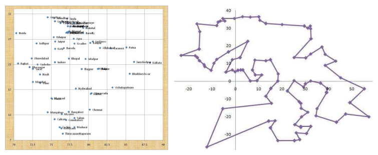


Figure 7: India 67 cities(our 1%), Africa and Islands(our 12%)

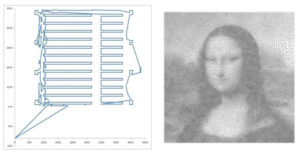


Figure 8: 2103 points PCB drilling(6%), Converting Pictures to Tours using Voronoi diagrams [33](2)

Algorithm 4 Transfer tour repair

```
while there exists nearby points on different segments do
    if hinge distance > crest distance i.e. $h_1 + h_2 + g_1 - g_2 - c_1 - c_2 > 0$ then
        Transfer points to nearer segment and decrease cost.
    end if
end while
```

Algorithm 5 Computing Dij from longitude and latitude [25]

```
PI = 3.141592. R=6378.388. /* Radius of earth*/
degree = (int) X[i]. minute = X[i] - degree.
radian = PI * (degree + 5 *minute/3)/180.
v1 = cos(lng[i] -lng[j]).
v2 = cos(lat[i] - lat[j]). v3 = cos(lat[i] + lat[j]).
Dij = (int) (R * acos(1/2 *((1 + v1)*v2 - (1 - v1)*v3))+1).
```

Miller-Tucker-Zemlin were among the first to provide formulations for TSP [14].
min $\sum_{i \in V} \sum_{j \in V, j>i} c_{ij} y_{ij}$ (minimize tour cost), Subject to,
$\sum_{j \in V, j>i} y_{ij} + \sum_{j \in V, j<i} y_{ji} = 2 \; \forall i \in V$ (vertex degree two),
$\sum_{i \in S} \sum_{j \in S, j>i} y_{ij} \leq |S| - 1 \; \forall \phi \neq S \subset V$ (no subtours),
$0 \leq y_{ij} \leq 1, \forall i, j \in V, j > i$, y_{ij} integer $\forall i, j \in V, j > i$.

We use the bounds obtained from the Held Karp lower bound [17, 18, 28], an LP relaxation, in Table 1 (see [25]). [4] uses in its backend linear programming solvers like CPLEX, Gurobi, Xpress solvers for solving the TSP problem.

Concorde solver developed by Robert Bixby, Vasek Chvatal, William Cook and David Applegate [7, 8], uses the cutting plane technique.

4.3 Held Karp Dynamic Programming

Algorithm 6, Held-Karp [15] dynamic programming is a $(n^2 2^n)$ time complexity algorithm for TSP. This memoizes the solutions to 2^n subsets of locations. Take some starting vertex s for the tour. For set of vertices R, $s \in R$, vertex $w \in R$, let $B(R, w)$ = minimum length of a path, starting in s visiting only all vertices in R and ending in w. Remembering the optimal subsolution (dynamic programming) for subsets reduces exponential term of the running time from $n!$ $((n/e)^n)$ to 2^n. It is a 50 year open problem if there is an exact algorithm for TSP with time (c^n) for $c < 2$ [27] (some recent progress has been made for cubic graphs [21, 20] and hamiltonian paths [19]). Memoization is popular in modern query optimizers including map reduce contexts [38].

Algorithm 6 Held Karp

```
$B(\{s\}, s) = 0$.
for all $S$ and $w$ and $|S| > 1$ do
    $B(S, w) = min_{v \in S - \{w\}} B(S - \{w\}, v) + weight(v, w)$.
end for
```

4.4 Christofides

Algorithm 7, Christofides's algorithm [16] is a 1.5 approximation to metric TSP. The MST (minimum spanning tree) is atmost the cost of $1 \times TSP$ as a TSP tour without a single edge is a spanning tree. A min weight matching is atmost $0.5 \times TSP$ as odd / even edges in a TSP tour give a matching. In practice 10-20% away from optimal solutions have been obtained [26]. It is a 35 year open problem if there is an approximation algorithm with factor < 1.5 (some recent progress has been made at Stanford for shortest path graph metrics [22, 23]). For the asymmetric case a similar algorithm recently developed by our colleagues at Stanford University achieves O(log n / loglog n) approximation [5].

Algorithm 7 Christofides

```
Get a MST T using Prim's or Kruskal's algorithm.
Set O = {v | v has odd degree in tree T}.
Compute a minimum weight matching M in the graph G[O].
Compute Euler tour C in graph T union M.
Add shortcuts to C to get a TSP-tour.
```

size	nn	nn-int	greedy	greedy-int
14	15.6	13.6	17	16.6
16	5.4	2.8	17.6	1.0
48	13	7.1	19.7	11.7
51	19.2	8.5	13	11
52	8.5	3.5	32.0	24.1
67	7.2	1.2	18.2	1
96	18.4	12.1	20.6	16.5
101	17	11.1	26.3	24.2
280	21.4	12.5	14.8	8.1
535	20.7	19.3	15.4	10.1
783	25	16.4	19.6	12.6
1002	21.4	13.6	19.2	14.4
2103	9.4	6.5		
14051	21.3	13.8		
33180	19.1	12.6		
85900	15.2	10.1		

Table 1: Performance of Excel Solver- %age away from optimal

4.5 Tours and Rectifications

Starting from size 33 instance in 1950s, the largest instance solved optimally till date is 85,900 locations taking 136 CPU years. Our results from Table 1 (for datasets from [25] except 67 in Figure 7) gives the percentage difference from optimal (obtained from Held Karp lower bound and [25]) of the solutions obtained from NN and greedy algorithms and with the intersection removal algorithm applied to the solutions. Greedy performs better on larger datasets but is more time expensive.

4.6 Metaheuristics

We also experimentally implemented heuristics like Simulated Annealing (SA)[31], Ant Colony Optimization (ACO)[30] and ElectroMagnetism(EM) like algorithm [32] for the TSP Problem whose results are shown in Table 2. Their complicated expensive noncombinatorial iteration rules lead to poor performance in CPU, RAM and approximation ratio especially as instance sizes increase.

size	EM	SA	ACO
14	15.0	18.4	15.0
52	8.5	17.2	6.5
96	18.2	35.9	14.2
159	15.4	29.5	14.3
226	15.9	17.6	13.1
299	20.2	27.9	20.8
654	24.2	28.3	24.0

Table 2: Performance of Metaheuristics- %age away from optimal

4.7 SQL Workload

In the first experiment, we generated 5 workloads with 100 queries each, each query a join of a random subset of 20 tables. Distance between two queries (with sets of tables $\Re_1$ and $\Re_2$) is the cardinality of the symmetric difference of the sets of tables in each queries join ($|\Re_1 \Delta \Re_2|$). This allows shared pipelined table scans and LRU RAM reuse. On an average across workloads, we observed the schedule developed by NN to be 3.7%, and greedy to be 2.9% away from optimal. In the second experiment we generated 5 workloads with 1000 queries each, each query selecting each table from totally 100 tables with probability 0.2 to be in the query's join (each approximately a 20 table join). On average 9.7 tables were shared between adjacent queries in the optimal ordering. The schedule developed by NN was 3.6% and greedy 2.3% away from optimal on average with 8.8 tables shared between adjacent queries compared to a random ordering that could achieve only four tables shared between adjacent queries. Considering columnar storages and cache policies, in a third experiment we considered a real world SAP workload containing 924 queries which reference on average 7.4 columns per query. The reordering increased the number of columns shared between adjacent queries from 0.42 to 4.9 on average. In a fourth experiment, a real world SAP workload of 16000 queries with on average 13.8 columns per query had originally 1.8 columns shared between adjacent queries already showing affinity, and after reordering shared 13.1 columns between adjacent queries, most being with same prepared statement template groupings. Template groupings make, batch execution techniques like JDBC rewrite [2], and cache reuse techniques [34, 35] that use LRU algorithm and time based aging across foreign keys, possible.

4.8 Critique of work

The most recent excel TSP solver [4] could solve upto 180 cities without running out of memory or time. We present a solver that can solve instances of upto 85,900 cities the largest instance solved optimally to date, approximately. With no extra software installation and a click of a button we are able to solve multiple different large sized TSP problems and provide tour rectifications for ordering problems. We provide an understanding of TSP solvers and show extensive testing on benchmark ordering problem datasets [25]. NEOS solver requires expensive dedicated servers [29].

Acknowledgements: Metaheuristics were developed with help from Prem Nathan, Prashant Kumar and Sani Kumbhar. Dr. Maitreya Natu provided the SAP workload. Dr. Rajiv Raman provided a few recent references.

5. REFERENCES

[1] S. Sudarshan, A. A. Diwan, Dilys Thomas, Scheduling and caching in multiquery optimization, COMAD 2006, 150–153.

[2] M. Chavan, R. Guravannavar, K. Ramachandra, S. Sudarshan, DBridge: A program rewrite tool for set-oriented query execution, ICDE 2011.

[3] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobhe. Efficient and extensible algorithms for multi-query optimization, SIGMOD 2000, 249–260.

[4] Rasmus Rasmussen, TSP in spreadsheets: A fast and flexible tool, Elsevier, Omega 39, 1, 51–63, January 2011.

[5] Arash Asadpour, Michel Goemans, Aleksander Madry, Shayan Oveis Gharan, Amin Saberi, An O(log n / log log n)-approximation algorithm for the asymmetric travelling salesman problem, SODA 2010.

[6] Donald Davendra, Traveling Salesman Problem, Theory and Applications, URL: http://www.intechopen.com, December 2010.

[7] Vasek Chvatal, Robert Bixby, William Cook, David Applegate, Traveling salesman problem: A computational study, PUP, 2006.

[8] D. Applegate, R. Bixby, V. Chvatal, and W. Cook, Concorde, TSP Solver, URL: http://www.tsp.gatech.edu/concorde/, 2006.

[9] Sanjeev Arora, Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems, JACM, 1998, 45, 5.

[10] Mitchell, J. S. B., Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems, SIAM Journal on Computing, 1999.

[11] Stephen Cook, The complexity of theorem proving procedures, STOC 1971, 151-158.

[12] Richard Karp, Reducibility among combinatorial problems, Complexity of Computer Computations, 1972, 85-103.

[13] J. Bang-Jensen, G. Gutin, A.Yeo, When the greedy algorithm fails, Discrete Optimization 1, 2004, 121-127.

[14] C.E. Miller, A.W. Tucker, R.A. Zemlin, Integer programming formulations and traveling salesman problems, JACM, 7, 1960, 326–329.

[15] M. Held, R. Karp. A dynamic programming approach to sequencing problems, Journal of SIAM, 1962, 10, 196-210.

[16] Nicos Christofides, Worst-case analysis of a new heuristic for the traveling salesman problem, Report 388, GSIA, CMU, 1976.

[17] M. Held, R. M. Karp, The traveling-salesman problem and minimum spanning trees, Operations Res. 18, 1970, 1138-1162.

[18] M. Held, R. M. Karp, The traveling-salesman problem and minimum spanning trees: Part II, Math. Programming 1, 1971, 6-25.

[19] Andreas Björklund, Determinant Sums for Undirected Hamiltonicity, FOCS 2010.

[20] Kazuo Iwama, Takuya Nakashima, An Improved Exact Algorithm for Cubic Graph TSP, COCOON 2007.

[21] David Eppstein, The Traveling Salesman Problem for Cubic Graphs, Journal of Graph Algorithms and Applications, 2007, 11(1) 61-81 .

[22] Shayan Oveis Gharan, Amin Saberi, Mohit Singh, A Randomized Rounding Approach to the Traveling Salesman Problem, FOCS 2011.

[23] Tobias Mömke, Ola Svensson, Approximating Graphic TSP by Matchings, FOCS 2011.

[24] S. Lin, B. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. Operations Research, 1973, 21(2), 498-516.

[25] G. Reinelt. TSPLIB. Universität Heidelberg, Institüt für Informatik, Im Neuenheimer Feld 368,D-69120 Heidelberg, Germany, 2004. URL http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

[26] M. Jünger, G. Reinelt, G. Rinaldi, The travelling salesman problem, Handbooks in Operations Res. & Management Sc., Elsevier, 1995.

[27] Gerhard Woeginger, Exact algorithms for NP-Hard problems, A survey, Combinatorial Optimization 2001, 185-208.

[28] D. S. Johnson, L. A. McGeoch, E. E. Rothberg, Asymptotic experimental analysis for the Held-Karp traveling salesman

bound, SODA, 1996.

[29] NEOS Server for Optimization, http://neos-server.org/neos/

[30] Marco Dorigo, Luca Maria Gambardella, Ant colonies for the traveling salesman problem, BioSystems 1997.

[31] S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi, Optimization by simulated annealing, Science, May 1983.

[32] S. Ilker Birbil, Shu-Cherng Fang, Electromagnetism-like mechanism for global optimization, Journal of Global Optimization, 2003, 25, 263-282.

[33] Robert Bosch, Opt Art, Math Horizons, February 2006, 14(3), 6–9.

[34] Times-Ten Team: Mid-tier caching: the TimesTen approach, (Now Oracle cache and in memory database), SIGMOD 2002, 588–593.

[35] SAP HANA, Realtime in memory technology, http://www.sap.com/hana/demos/index.epx

[36] G. Graefe and W. J. McKenna, Extensibility and search efficiency in the Volcano optimizer generator, ICDE, 1993, 209–218.

[37] P. G. Selinger, M.M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. Price, Access path selection in relational database management system, In ACM SIGMOD Intl. Conf. Management of Data, 1979, 23–34.

[38] Foto N. Afrati, Jeffrey D. Ullman, Optimizing Multiway Joins in a Map-Reduce Environment, IEEE TKDE 2011, 23(9): 1282-1298

[39] M. Hong, M. Riedewald, C. Koch, J. Gehrke, and A. Demers, Rule-Based Multi-Query Optimization, EDBT, 120–131, 2009

[40] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, An analysis of several heuristics for the traveling salesman problem, SICOMP 563–581, 1977.

W O R K   I N   P R O G R E S S

A lightweight distributed order and duplication insensitive algorithm for approximate top-k queries using order statistics

Vinay Deolalikar
Hewlett Packard Labs
1501 Page Mill Road
Palo Alto, CA 94304
vinayd@hpl.hp.com

Kave Eshghi
Hewlett Packard Labs
1501 Page Mill Road
Palo Alto, CA 94304
kave@hpl.hp.com

Hernan Laffitte
Hewlett Packard Labs
1501 Page Mill Road
Palo Alto, CA 94304
hernan@hpl.hp.com

1. APPROXIMATE TOP-K

Let $\{e_1, e_2, \ldots, e_l\}$ be a set of distinct records in a database, with unique IDs $\{id_1, id_2, \ldots, id_l\}$. Let $A_1, A_2, \ldots, A_p$ be a set of distinct attributes for each record. For every record e_i, the attribute A_j is zero or some positive value. We denote the value of the attribute A_j of record e_i by $A_j(e_i)$. The sum of the attributes of e_i is denoted by $N_i = \sum_j A_j(e_i)$. We would like to obtain the list of top k records, ordered by N_i. We present a highly configurable, lightweight, distributed algorithm to solve the above problem approximately, based on order statistics.

2. THE ALGORITHM

2.1 Phase One: Generating a list of random variables

A ticket is a triple $< ID, r, b >$ where id is a record, r is the value of a random variable, and b is a binary flag which can be set to either 1 or 0, respectively.

Each peer first generates an exponential random variable for the record e_i with rate given by $A_j(e_i)$. At the end of this phase, each peer will have a list of random variables that is as long as the number of records. The list has two columns: the first column has the record ID and the second column has the random variable value.

2.2 Phase Two: Pruning the list

Each peer thresholds the list of random variables they have generated. Rows in the list whose second column (random variable value) is below a threshold T are discarded.

2.3 Phase Three: Exchanging lists

In the third phase of the algorithm, each peer sends their pruned list of (record ID, random variable value) to their neighbors. This is the information passing phase of the algorithm.

2.4 Phase Four: Merging lists by maximum

Each peer now has lists from other peers. Each peer now merges these lists by keeping only the maximum of the values of the random variables for each record.

2.5 Phase Five: Cropping merged lists

Each peer now sorts his merged list in descending order of random variable value, and crops it to have only L topmost records.

Now the algorithm proceeds by looping through phases Two through Five for a fixed number of iterations. Experimental results indicate that 5 iterations suffice for a stabilization of lists.

2.6 Phase Six: Running algorithm multiple times and merging results

Phases One through Six are run a 'run count' of times. At the end of each run, a list emerges. Now, a final list is obtained as follows. If a record occurs in at least 'merge count' out of the total 'run count' number of lists, then it is included in the final output of the algorithm as a top k record.

As with any approximate algorithm, we may merge results of multiple runs of the basic algorithm outlined above, in order to increase accuracy.

3. EXPERIMENTAL RESULTS

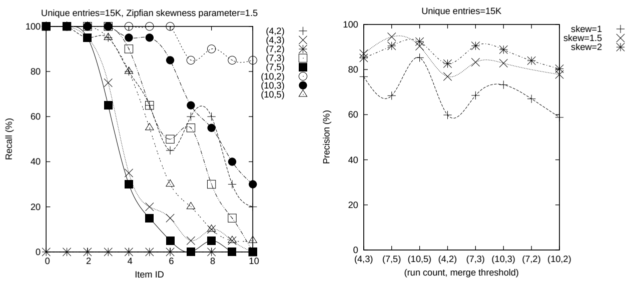


Figure 1: Tradeoff between (run count, merge threshold) and precision vs. recall on Zipfian distribution over 15K distincts. Item id indicates frequency of item: Item 0 is most frequent, and so on.

We have validated our algorithm extensively on a wide array of multi-parameter Zipfian datasets, varying the skewness of the distribution of records (only one parameter choice shown in Fig. 1). We report strong performance of the algorithm over a wide range of parameter values, and study the trade-offs involved in setting the tunable parameters of the algorithm in order to obtain the precision and recall that is desired.

Who's Who: Linking User's Multiple Identities on Online Social Media

Paridhi Jain
IIIT-Delhi, India
paridhij@iiitd.ac.in

Ponnurangam Kumaraguru
IIIT-Delhi, India
pk@iiitd.ac.in

Anupam Joshi
UMBC, United States
joshi@cs.umbc.edu

1. ABSTRACT

On online social media, users join new online social networks (OSNs) to exploit variety of services while maintaining their old identities on other OSNs. A user maintains an identity on each OSN mentioning metadata (e.g. profile information) about her. Heterogeneity of metadata shared by user across OSNs leads to a problem of finding if two online identities on multiple OSNs belong to the same user or different users. In this work, we attempt to understand that to what extent can we link multiple online identities of a user or disambiguate identities of different users, using an easily accessible and public attribute – username. The solution to the problem has multiple applications. In privacy domain, the problem finds its application in understanding the quantity and quality of the user's information leakages via either aggregation of user's information from OSNs or differences in privacy policies of multiple social networks. In system building domain, the solution can help in building recommendation feature for social aggregation sites. In security domain, the solution can help in linking malicious user accounts present on multiple OSNs.

1.1 Methodology

We collected usernames of 1,193 users on different social networks and created two datasets by different methods. In dataset 1, no two users shared the same name and hence their usernames were distinct and easily separable. However in real world, disambiguation of two users with similar names was a challenge. Therefore in dataset 2, there exists users who shared the same first name and had similar usernames. The existence of similar usernames belonging to different users challenged the techniques we proposed to link identities of a user and disambiguate identities of different users.

We proposed a set of string based features to capture the possible similarities a user's two usernames had, in order to predict if two usernames belong to the same user. Some of the strong features were – n-gram coefficient, Jaccard coefficient, Affine gap and Smith-Waterman distance.

1.2 Analysis and Results

We analyzed 1,193 users and found that 359 users (30%) used same username, and 327 (27.4%) users had twisted versions of the user's most used username on every OSN. Rest of the users had atleast one different username on at least one OSN. We observed that a more than half the users (30% + 27.4%) had same or similar users, with possible reasons as – the username they wished was already taken or they modified their username on the basis of social network nature. This motivated the string based feature set and the techniques we discuss to link users.

1.2.1 Classification

We extracted a set of string based features for a username pair (437,836 pairs for dataset 1 and 4,384 pairs for dataset 2) either belonging to same user or different users. We performed random sub-sampling validation 10 times with 50% training and 50% testing dataset. We used SVM with RBF kernel to classify the username pair if it belonged to the same user or two different users. SVM with the training accuracy of 93.7% for dataset 1 and 85% for dataset 2, yielded a classification accuracy of 99.85% on dataset 1 while 75.37% on dataset 2.

The classification accuracy (99.85%) is higher than the state-of-the-art accuracy (71%) by Perito et. al [1] which experiments with 10,000 username pairs of the users where no two users have same names (similar to dataset 1). The higher accuracy shows that string based features are efficient in predicting similarities of two usernames of a user. However, the classifier makes errors when different users with similar usernames are marked as the usernames belong to the same user (accuracy - 75.37%). To distinguish between users with similar name, we need to incorporate other attributes e.g. profile and network attributes.

1.3 Conclusion

In conclusion, we observe that majority (57.4%) of users use same or similar usernames across multiple online networks. Therefore we argue that username can be used as a unique identifier to link user identities across OSNs. With string based features of a username pair, accuracy of correct prediction can be improved from 71% to 99.85%.

2. REFERENCES

[1] D. Perito, C. Castelluccia, M. A. Kâafar, and P. Manils, "How unique and traceable are usernames?" in *PETS*, 2011.

MODETL: A complete MODeling and ETL method for designing Data Warehouses from Semantic Databases

Selma Khouri
LIAS/ISAE-ENSMA
France
selma.khouri@ensma.fr

Ladjel Bellatreche
LIAS/ISAE-ENSMA
France
bellatreche@ensma.fr

Nabila Berkani
National High School for
Computer Science, Algeria
n_berkani@esi.dz

ABSTRACT

In last decades, Semantic DataBases ($\mathcal{SDB}$) have emerged and the major DBMS editors provide semantic support in their products. This is mainly due to the spectacular development of ontologies in several important domains like E-commerce, Engineering, Medicine, etc. Note that ontologies can be seen as a natural continuity of conceptual models. Contrary to traditional databases, where their instances are stored in a relational layout, $\mathcal{SDB}$ store ontological data according to one of three main storage layouts (horizontal, vertical, binary). Actually, $\mathcal{SDB}$ are serious candidates for business intelligence applications built around the Data Warehouse ($\mathcal{DW}$) technology. The important steps of the life-cycle warehouse design (user requirement analysis, conceptual design, logical design, ETL process, physical design) are usually managed in isolation way. This treatment is mainly due to the complexity of each phase. Actually, $\mathcal{DW}$ technology is quite mature for traditional data sources. As a consequence, leveraging its steps to deal with $\mathcal{SDB}$ becomes a necessity. In this paper, we propose a method that covers the most important steps of life-cycle of semantic $\mathcal{DW}$. To fitful our needs, four main objectives have been defined:

O_1: leveraging the integration framework by considering ontologies: a $\mathcal{DW}$ can be seen as a materialized data integration system, where data are viewed in a multidimensional way. Data integration systems are formally defined by a triple: $<\mathcal{G}, \mathcal{S}, \mathcal{M}>$, where $\mathcal{G}$ is the global schema, $\mathcal{S}$ is a set of local schemas that describes the structure of each source participating in the integration process, and $\mathcal{M}$ is a set of assertions relating elements of the global schema $\mathcal{G}$ with elements of local schemas $\mathcal{S}$. We defined an integration framework $<\mathcal{G}, \mathcal{S}, \mathcal{M}>$ adapted to $\mathcal{SDB}$ specificities, where schema $\mathcal{G}$ is represented by a domain ontology, the set of sources S considered are $\mathcal{SDB}$, and M represents the set of mapping assertions. A mathematical formalization of ontologies, $\mathcal{SDB}$ and semantic $\mathcal{DW}$ is given, based on the description logic formalism.

O_2: User requirements have to be expressed at the ontological level: the requirements model we proposed follows the *goal oriented paradigm*. After analyzing the major studies related to this paradigm, we proposed a goal model viewed as a pivot model, since it combines three widespread goal-oriented approaches: *KAOS, Tropos* and *iStar*. The goal model is then connected to the ontology meta-model in order to specify requirements at the ontological level. Requirements analysis allows the designer to construct the dictionary identifying the set of relevant concepts and properties used by the target application. The conceptual, logical and then physical model are defined based on that dictionary. The availability of the ontology allows exploiting its reasoning capabilities to correct the inconsistencies of the conceptual model, and to infer new facts.

O_3: The ETL process has to be defined at the ontological level and not at physical or conceptual levels: different ETL works proposed in the literature consider logical schemas of sources as inputs of the $\mathcal{DW}$ system, and make an implicit assumption that the $\mathcal{DW}$ model will be deployed using the same representation (usually using a relational representation). The third objective of our method ensures the definition of the ETL process at the ontological level independently of any implementation constraint. We defined a generic ETL algorithm, based on ten generic operators defined in the literature, that aims at populating the target $\mathcal{DW}$ schema, by data from $\mathcal{SDB}$.

O_4: the deployment process needs to consider the different storage layouts of semantic $\mathcal{DW}$: different deployment solutions are proposed and implemented using data access object design patterns. A prototype validating our proposal using the Lehigh University Benchmark ontology and Oracle $\mathcal{SDB}$ has been developed.

Categories and Subject Descriptors

H.2.7 [**Database management**]: [Data warehouse and repository]; D.2.10 [**Software engineering**]: Design—*Methodologies*

Keywords

Data warehouse design, Ontology, Semantic databases, ETL process

Web Personalization and Recommender Systems: An Overview

R. B. Wagh[*]
Research Scholar, Dept.of Computer Engineering, RCPIT
Shirpur, Maharashtra, India
rajnikantw@gmail.com

Prof. Dr. J. B. Patil[†]
Principal & Professor, Dept.of Computer Engineering, RCPIT
Shirpur, Maharashtra, India
jbpatil@hotmail.com

ABSTRACT

Information overload is the major problem of today's Internet use. User frequently gets much more information than needed. Also much of the information which the user gets is less relevant and very few links, items, or contents are really useful. To get rid of this problem, Web Personalization or Recommender System is widely used now. It aims at fulfilling the user needs more appropriately. By analyzing and mining Web content data, structure data, usage data and user profile data, system achieves the goal of user satisfaction. In this paper, we focus on existing methods, their mechanism, limitations and possible extensions which may improve the capabilities. In our proposed work, we will improve the accuracy of recommender system. For this purpose, we will make use of various classification and clustering methods. Presently we are concentrating on density based, hierarchical and message passing algorithms to achieve the desired goal of accuracy. More specifically, our aim is to show that graph based message passing algorithms may outperform than K-means algorithm which makes use of partition method. The methodology used for recommendation purpose will be based on collaborative filtering approach. Presently we are working on log file of our engineering colleges' web site namely `www.rcpit.ac.in.` Our aim is to analyze user behaviour in terms of navigational paths and to recommend them the future navigations to help achieve the necessary data in less time. Since the present log file is not much larger and also the navigational patterns are also less or alike, we are trying to get the dataset of North Maharashtra University, Jalgaon web site, namely `www.nmu.ac.in`. Also, we will make use of some standard datasets like CTI, MSNBC, Grouplens or Netflix for our experiment and evaluation purpose. We will use above mentioned clustering and classification techniques to improve browsing experience of user.

[*]R. B. Wagh
[†]Prof. Dr. J. B. Patil

The 18th International Conference on Management of Data (COMAD), 14th-16th Dec, 2012 at Pune, India.

Keywords- Web personalization; Recommender system; Web content data; Web structure data, Web usage data.

1. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005.
[2] M. Eirinaki and M. Vazirgiannis. Web mining for web personalization. *ACM Trans. Internet Technol.*, 3(1):1–27, Feb. 2003.
[3] B. K. F.Ricci, L.Rokac. *Recommender Systems Handbook*. 2011.
[4] M. Jalali, N. Mustapha, M. N. Sulaiman, and A. Mamat. Webpum: A web-based recommendation system to predict user future movements. *Expert Syst. Appl.*, 37(9):6201–6212, sep 2010.
[5] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. Artificial Intellegence*, 2009.
[6] N. M. Yahya AlMurtadha, Md. Nasir Bin Sulaiman and N. I. Udzir. Ipact: Improved web page recommendation system using profile aggregation based on clustering of transactions. *American Journal of Applied Sciences*, 8(3):277–283, 2011.

Efficient Approximate Dictionary Matching

Saurabh Kishore
skishore@gmail.com

Ashish V. Tendulkar
ashishvt@gmail.com

ABSTRACT

Named entity recognition (NER) systems are important for extracting useful information from unstructured data sources. It is known that large domain dictionaries help in improving extraction performance of NER. Unstructured text usually contains entity mentions that are different from their standard dictionary form. Approximate matching is important to identify the correct dictionary entity for such variants. This is a challenging problem, as every entity in the dictionary is a candidate match for the variant. In this paper, we propose a novel approach for efficient approximate dictionary matching. The key idea is to compare a given query only against a set of most likely candidate matches from the dictionary so as to achieve substantial reduction in the number of matching operations. In order to enable this, the proposed approach first performs clustering of similar entities and then represents each cluster with a profile matrix, which stores the probability of an occurrence of a particular character at a specific location in the entity string. Thus, the dictionary is represented with a set of profile matrices, which are much smaller than the actual number of entities. A given query entity is first matched against the profiles and the clusters corresponding to top-K best scoring profiles are selected to obtain a list of most likely matching candidates. The query is then compared with each candidate match entity and the approximate match is declared if both the query and the candidate entity are within acceptable edit distance threshold. We have performed rigorous evaluation of our approach on several publicly available datasets. The proposed algorithm outperforms alternative approaches in detecting approximately matching entities for a given query using far lesser number of comparison operations.