Sang-goo Lee Zhiyong Peng
Xiaofang Zhou Yang-Sae Moon
Rainer Unland Jaesoo Yoo (Eds.)

LNCS 7238

Database Systems for Advanced Applications

17th International Conference, DASFAA 2012
Busan, South Korea, April 2012
Proceedings, Part I

1 Part I

Springer

Lecture Notes in Computer Science 7238

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison
Lancaster University, UK

Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler
University of Surrey, Guildford, UK

Jon M. Kleinberg
Cornell University, Ithaca, NY, USA

Alfred Kobsa
University of California, Irvine, CA, USA

Friedemann Mattern
ETH Zurich, Switzerland

John C. Mitchell
Stanford University, CA, USA

Moni Naor
Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz
University of Bern, Switzerland

C. Pandu Rangan
Indian Institute of Technology, Madras, India

Bernhard Steffen
TU Dortmund University, Germany

Madhu Sudan
Microsoft Research, Cambridge, MA, USA

Demetri Terzopoulos
University of California, Los Angeles, CA, USA

Doug Tygar
University of California, Berkeley, CA, USA

Gerhard Weikum
Max Planck Institute for Informatics, Saarbruecken, Germany

Sang-goo Lee Zhiyong Peng
Xiaofang Zhou Yang-Sae Moon
Rainer Unland Jaesoo Yoo (Eds.)

Database Systems for Advanced Applications

17th International Conference, DASFAA 2012
Busan, South Korea, April 15-18, 2012
Proceedings, Part I

Springer

Volume Editors

Sang-goo Lee
Seoul National University, Seoul 151747, South Korea
E-mail: sglee@snu.ac.kr

Zhiyong Peng
Wuhan University, Wuhan 430081, Hubei Province, China
E-mail: peng@whu.edu.cn

Xiaofang Zhou
University of Queensland, Brisbane, QLD 4072, Australia
E-mail: zxf@itee.uq.edu.au

Yang-Sae Moon
Kangwon National University, Chuncheon 200701, Kangwon, South Korea
E-mail: ysmoon@kangwon.ac.kr

Rainer Unland
University of Duisburg-Essen, 45117 Essen, Germany
E-mail: rainer.unland@icb.uni-due.de

Jaesoo Yoo
Chungbuk National University, Cheongju 361-763, Chungbuk, South Korea
E-mail: yjs@chungbuk.ac.kr

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-29037-4 e-ISBN 978-3-642-29038-1
DOI 10.1007/978-3-642-29038-1
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012933786

CR Subject Classification (1998): H.2-5, C.2, J.1, J.3

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

© Springer-Verlag Berlin Heidelberg 2012
This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.
The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

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

Printed on acid-free paper

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

Preface

It is our great pleasure to welcome you to the proceedings of the 17th International Conference on Database Systems for Advanced Applications (DASFAA 2012), which was held in Busan, Korea, in April, 2012. DASFAA is an international conference which provides a forum for technical presentations and discussions among database researchers, developers and users from academia, business and industry, in the areas of databases, large-scale data management, data mining, search and recommendation, and the Web.

The call for papers attracted 159 research submissions from 24 countries (based on the affiliation of the first author). Among them, the Program Committee selected, through a comprehensive review process, 44 regular papers and 8 short papers for presentation. The Industrial Committee, chaired by Won Suk Lee, Mukesh Mohania and Jeffrey Yu, selected 8 industrial papers for presentation. The conference program also included 8 demo presentations selected from 17 submissions by the Demo Committee chaired by Wolf-Tilo Balke and Seung-Won Hwang.

This volume also includes extended abstracts of the two invited keynote lectures by Divesh Srivastava (AT&T Research) and Sang Kyun Cha (Seoul National University and SAP Labs Korea), whose topics were on "Enabling Real-Time Data Analysis" and "A New Paradigm of Thinking and Architecture for Real-Time Information Processing at Fingertips," respectively. The Tutorial Chair, Wook-Shin Han, organized four tutorials by leading experts on topics ranging from probabilistic databases to detecting clones and reuse on the Web. A stimulating panel was organized by the Panel Chair, Kyuseok Shim. This rich and attractive conference program boasts conference proceedings that span two volumes of Springer's *Lecture Notes in Computer Science* series.

Beyond the main conference Hwanjo Yu, Yu Ge and Wynne Hsu, who chaired the Workshop Committee, put together five workshops that catered to specific interests of the conference participants. The workshop papers are included in a separate volume of proceedings also published by Springer in its *Lecture Notes in Computer Science* series.

DASFAA 2012 was jointly organized by Pusan National University and the Database Society of Korea. It received in-cooperation sponsorship from the Korea Institute of Information Scientists and Engineers, the Database Society of Japan, the China Computer Federation Database Technical Committee, and the Korea Database Agency. We are grateful to the industry and institutional sponsors who contributed generously to making DASFAA 2012 successful.

The conference would not have been possible without the support and hard work of many colleagues. We would like to express our special thanks to Honorary Conference Chair, Kyu-Young Whang, for his valuable advice on all aspects of organizing the conference. We thank the DASFAA Steering Committee for their leaderships and encouragement. We thank the General Co-chairs, Yoon Joon Lee and Kazutoshi Sumiya, Organizing Committee Chair, Bonghee Hong, Publicity Co-chairs, Eenjun Hwang, Jae-Gil Lee and YunChan Chang, Local Arrangements Committee Co-chairs, Joonho Kwon and Ok-Ran Jeong, Finance Chair, Min-Su Lee, Web Co-chairs, Ha-Joo Song and Young-Koo Lee, Demo Award Committee Co-chairs, Young-Kuk Kim, Takahiro Hara and Kyoung-Gu Woo, Best Paper Committee Co-chairs, SangKeun Lee, Hiroyuki Kitagawa and Xiaofeng Meng, Sponsor Co-chairs, Yunmook Nah and Kyu-Chul Lee, Registration Chair, Sanghyun Park, Steering Committee Liaison, Byeong-Soo Jeong, APWEB Liaison, Wookey Lee, and EDB Liason, Jinho Kim.

Finally, our thanks go to all the committee members and other individuals involved in putting this all together, and to all authors who submitted their papers to this conference.

April 2012

Sang-goo Lee
Zhiyong Peng
Xiaofang Zhou
Yang-Sae Moon
Rainer Unland
Jaesoo Yoo

Organization

Honorary Conference Chair

Kyu-Young Whang	KAIST, South Korea

Conference General Co-chairs

Yoon Joon Lee	KAIST, South Korea
Kazutoshi Sumiya	University of Hyogo, Japan

Program Committee Co-chairs

Sang-goo Lee	Seoul National University, South Korea
Zhiyong Peng	Wuhan University, China
Xiaofang Zhou	University of Queensland, Australia

Organizing Committee Chair

Bonghee Hong	Pusan National University, South Korea

Workshop Co-chairs

Hwanjo Yu	POSTECH, South Korea
Yu Ge	Northeastern University, China
Wynne Hsu	National University of Singapore, Singapore

Industrial Co-chairs

Won Suk Lee	Yonsei University, South Korea
Mukesh K. Mohania	IBM Research, India
Jeffrey Xu Yu	Chinese University of Hong Kong, China

Tutorial Chair

Wook-Shin Han	Kyungbook National University, South Korea

Panel Chair

Kyuseok Shim	Seoul National University, South Korea

Demo Co-chairs

Wolf-Tilo Balke	TU-Braunschweig, Germany
Seung-Won Hwang	POSTECH, South Korea

Publicity Co-chairs

Eenjun Hwang	Korea University, South Korea
Jae-Gil Lee	KAIST, South Korea
YunChan Chang	Victoria University, Australia

Local Arrangements Co-chairs

Joonho Kwon	Pusan National University, South Korea
Ok-Ran Jeong	Gachon University, South Korea

Finance Chair

Min-Su Lee	Ewha Womans University, South Korea

Publication Co-chairs

Rainer Unland	University of Duisburg-Essen, Germany
Jaesoo Yoo	Chungbuk National University, South Korea
Yang-Sae Moon	Kangwon National University, South Korea

Web Co-chairs

Ha-Joo Song	Pukyong National University, South Korea
Young-Koo Lee	Kyung Hee University, South Korea

Demo Award Committee Co-chairs

Young-Kuk Kim	Chungnam National University, South Korea
Takahiro Hara	Osaka University, Japan
Kyoung-Gu Woo	Samsung Electronics, South Korea

Best Paper Committee Co-chairs

SangKeun Lee	Korea University, South Korea
Hiroyuki Kitagawa	University of Tsukuba, Japan
Xiaofeng Meng	Renmin University of China, China

Steering Committee Liaison

Byeong-Soo Jeong	Kyung Hee University, South Korea

Sponsor Co-chairs

Yunmook Nah	Dankook University, South Korea
Kyu-Chul Lee	Chungnam National University, South Korea

Registration Chair

Sanghyun Park	Yonsei University, South Korea

APWEB Liaison

Wookey Lee	Inha University, South Korea

EDB (International Conference on Emerging Databases) Liaison

Jinho Kim	Kangwon National University, South Korea

DASFAA Steering Committee

Ramamohanarao Kotagiri (Chair)	University of Melbourne, Australia
Jianzhong Li (Vice Chair)	Harbin Institute of Technology, China
Katsumi Tanaka (Advisor)	Kyoto University, Japan
Kazutoshi Sumiya (Treasurer)	University of Hyogo, Japan
Qing Li (Secretary)	City University of Hong Kong, China
Masaru Kitsuregawa	University of Tokyo, Japan
Mukesh K. Mohania	IBM Research, India
Byeong-Soo Jeong	Kyung Hee University, South Korea
Ming-Syan Chen	National Taiwan University, Taiwan
Eui Kyeong Hong	University of Seoul, South Korea
Hiroyuki Kitagawa	University of Tsukuba, Japan
Li-Zhu Zhou	Tsinghua University, China
Stephane Bressan	National University of Singapore, Singapore
BongHee Hong	Pusan National University, South Korea

Program Committees

Research Track

Toshiyuki Amagasa	University of Tsukuba, Japan
Masayoshi Aritsugi	Kumamoto University, Japan
Zhifeng Bao	National University of Singapore, Singapore
Ladjel Bellatreche	Poitiers University, France
Boualem Benatallah	University of New South Wales, Australia
Sourav Bhowmick	Nanyang Technological University, Singapore
Cui Bin	Peking University, China
Athman Bouguettaya	RMIT, Australia
Jinseok Chae	University of Incheon, South Korea
Chee Yong Chan	National University of Singapore, Singapore
Jae Woo Chang	Chonbuk National University, South Korea
Jae-young Chang	Hansung University, South Korea
Lei Chen	HKUST, China
Ming-Syan Chen	National Taiwan University, Taiwan
Yi Chen	Arizona State University, USA
Hong Cheng	Chinese University of Hong Kong, China
James Cheng	Nanyang Technological University, Singapore
Reynold Cheng	University of Hong Kong, China
Jae-heon Cheong	Shingu University, South Korea
Byron Choi	Hong Kong Baptist University, China
Yon Dohn Chung	Korea University, South Korea
Gao Cong	Nanyang Technological University, Singapore
Alfredo Cuzzocrea	ICAR-CNR / University of Calabria, Italy
Gill Dobbie	University of Auckland, New Zealand
Xiaoyong Du	Renmin University of China, China
Jianhua Feng	Tsinghua University, China
Ling Feng	Tsinghua University, China
Yunjun Gao	Zhejiang University, China
Yu Ge	Northeastern University, China
Stephane Grumbach	INRIA, France
Takahiro Hara	Osaka University, Japan
Bingsheng He	Nanyang Technological University, Singapore
Wynne Hsu	National University of Singapore, Singapore
Haibo Hu	Hong Kong Baptist University, China
Ming Hua	Facebook, USA
Dong-Hyuk Im	Seoul National University, South Korea
Yoshiharu Ishikawa	Nagoya University, Japan
Adam Jatowt	Kyoto University, Japan
Ruoming Jin	Kent State University, USA
Sungwon Jung	Sogang University, South Korea

Norio Katayama	National Institute of Informatics, Japan
Yiping Ke	Chinese University of Hong Kong, China
Chulyon Kim	Kyungwon University, South Korea
Dongkyu Kim	Georgetown University, USA
Han-joon Kim	University of Seoul, South Korea
Jinho Kim	Kangwon National University, South Korea
Sang-Wook Kim	Hanyang University, South Korea
Markus Kirchberg	HP Labs Singapore, Singapore
Hiroyuki Kitagawa	University of Tsukuba, Japan
Ig-hoon Lee	Seoul National University, South Korea
Mong Li Lee	National University of Singapore, Singapore
Sang-Won Lee	Sungkyunkwan University, South Korea
Wang-Chien Lee	Pennsylvania State University, USA
Cuiping Li	Renmin University of China, China
Jianzhong Li	Harbin Institute of Technology, China
Xuemin Lin	University of New South Wales, Australia
Chengfei Liu	Swinburne University of Technology, Australia
Eric Lo	Hong Kong Polytechnic University, China
Jiaheng Lu	Renmin University of China, China
Nikos Mamoulis	University of Hong Kong, China
Weiyi Meng	Binghamton University, USA
Xiaofeng Meng	Renmin University of China, China
Jun-Ki Min	Korea University of Technology and Education, South Korea
Jun Miyazaki	Nara Advanced Institute of Science and Technology, Japan
Bongki Moon	University of Arizona, USA
Yang-Sae Moon	Kangwon National University, South Korea
Yasuhiko Morimoto	Hiroshima University, Japan
Atsuyuki Morishima	University of Tsukuba, Japan
Miyuki Nakano	University of Tokyo, Japan
Tadashi Ohmori	University of Electro-Communications, Japan
Makoto Onizuka	NTT Corporation, Japan
Hyoungmin Park	University of Brithish Columbia, Canada
Min Sik Park	Korea Database Agency, South Korea
Sanghyun Park	Yonsei University, South Korea
Young-Ho Park	Sookmyung Women's University, South Korea
Jian Pei	Simon Fraser University, Canada
Wen-Chih Peng	National Chiao Tung University, Taiwan
Lu Qin	Chinese University of Hong Kong, China
Keun Ho Ryu	Chungbuk National University, South Korea
Simonas Saltenis	Aalborg University, Denmark
Markus Schneider	University of Florida, USA
Jialie Shen	Singapore Management University, Singapore

Junho Shim	Sookmyung Women's University, South Korea
Hyoseop Shin	Konkuk University, South Korea
Jung Hyeon Sin	INET-Hosting, South Korea
Atsuhiro Takasu	National Institute of Informatics, Japan
David Taniar	Monash University, Australia
Vincent Tseng	National Cheng Kung University, Taiwan
Haixun Wang	Microsoft Research Asia, China
Jianyong Wang	Tsinghua University, China
John Wang	Griffith University, Australia
Wei Wang	University of New South Wales, Australia
Raymond Wong	HKUST, China
Xiaokui Xiao	Nanyang Technological University, Singapore
Jianliang Xu	Hong Kong Baptist University, China
Ke Yi	HKUST, China
Man Lung Yiu	Hong Kong Polytechnic University, China
Haruo Yokota	Tokyo Institute of Technology, Japan
Jaesoo Yoo	Chungbuk National University, South Korea
Rui Zhang	University of Melbourne, Australia
Wenjie Zhang	University of New South Wales, Australia
Baihua Zheng	Singapore Management University, Singapore
Bin Zhou	University of Maryland Baltimore County, USA

Industrial Track

Haibo Hu	Hong Kong Baptist University, China
Weining Qian	Fudan University, China
Bingsheng HE	Nanyang Technological University, Singapore
Marek Kowalkiewicz	SAP, Australia
Jilei Tian	Nokia Research China, China
Unil Yun	Chungbuk National University, South Korea
Yang-Sae Moon	Kangwon National University, South Korea
Byungjoo Chung	Cubrid, South Korea

Demo Track

Ilaria Bartolini	University of Bologna, Italy
Changkyu Kim	Intel Labs, USA
Jiaheng Lu	Renmin University of China, China
Yaokai Feng	Kyushu University, Japan
Young-In Song	Microsoft Research Asia, China
Yoonkyong Lee	Samsung Electronics, South Korea
Georgia Koutrika	IBM Research, USA
Christoph Lofi	TU-Braunschweig, Germany

External Reviewers

Brian Ackerman
Kamel Boukhalfa
Panagiotis Bouros
Yulei Fan
Wei Feng
Shen Ge
Reza Hemayati
Hai Huang
Zheng Huo
Stéphane Jean
Yu Jiang
Selma Khouri
Chungrim Kim
Young-kook Kim
Ryan Ko
Erwin Leonardi
Jing Li
Jianxin Li
Wenxin Liang
Pan Lin
Lin Liu
Yunzhong Liu
Cheng Long
Youzhong Ma
Luo Min
Jaehui Park
Peng Peng
Yu Peng
Zhenhua Song
Yu Shyang Tan
Yongxin Tong
Guan Wang
Hao Wang
Yousuke Watanabe
Kefeng Xuan
MingxuanYuan
Geng Zhao
Jinzeng Zhang
Rui Zhou
Wei Zhang
Xiaojian Zhang

Table of Contents – Part I

Keynote Talks

Query Processing and Optimization

Data Semantics and Interoperability

XML and Semi-structured Data I

XML and Semi-structured Data II

Data Mining and Knowledge Discovery I

Data Mining and Knowledge Discovery II

Data Mining and Knowledge Discovery III

Privacy and Anonymity

Data Management in the Web

Graphs and Data Mining Applications

Temporal and Spatial Data I

Temporal and Spatial Data II

Table of Contents – Part II

Top-k and Skyline Query Processing

Information Retrieval and Recommendation

Indexing and Search Systems

Cloud Computing and Scalability

Industrial Papers I: Memory-Based Query Processing

Industrial Papers II: Semantic and Decision Support Systems

Demo Papers I: Social Data

Demo Papers II: Data Mining

Panel

Tutorials

Enabling Real Time Data Analysis

Divesh Srivastava

AT&T Labs-Research, Florham Park NJ 07932, USA
divesh@research.att.com

Abstract. Network-based services have become a ubiquitous part of our lives, to the point where individuals and businesses have often come to critically rely on them. Building and maintaining such reliable, high performance network and service infrastructures requires the ability to rapidly investigate and resolve complex service and performance impacting issues. To achieve this, it is important to collect, correlate and analyze massive amounts of data from a diverse collection of data sources in real time.

We have designed and implemented a variety of data systems at AT&T Labs-Research to build highly scalable databases that support real time data collection, correlation and analysis, including (a) the Daytona data management system, (b) the DataDepot data warehousing system, (c) the GS tool data stream management system, and (d) the Bistro data feed manager. Together, these data systems have enabled the creation and maintenance of a data warehouse and data analysis infrastructure for troubleshooting complex issues in the network. We describe these data systems and their key research contributions in this talk.

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, p. 1, 2012.
© Springer-Verlag Berlin Heidelberg 2012

A New Paradigm of Thinking and Architecture for Real-Time Information Processing at Fingertips

Sang Kyun Cha[1,2]

[1] Seoul National University
chask@snu.ac.kr
[2] SAP Labs Korea
sang.k.cha@sap.com

Today's enterprise-scale information systems comprise of complex vertical tiers of database, application, web, and mobile servers. Horizontal tiers of OLTP and OLAP systems add further complexity to enterprise information management. Historically, such introduction of vertical and horizontal tiers was inevitable to address the complexity and performance problems in the course of building up enterprise applications by divide and conquer. However, these tiers have accumulated so much redundancy and overhead over time, making the overall system difficult and expensive to maintain.

Over past decades, we have persistently observed exponential growth of hardware power following the well-known Moore's law. A commodity server can now have hundreds of cores and terabytes of memory, which were not conceivable other than in supercomputers several years ago, at a fraction of cost. This trend is likely to continue at least several years, and at least ten times of further increase of hardware processing power is expected in the near future.

The dramatic hardware advance has brought us to an inflection point that we can eliminate these complex tiers to streamline information delivery to the new generation of end users demanding real-time decision making at fingertips any time anywhere. SAP HANA platform was designed with this rethinking of tiers in enterprise-scale information systems, leveraging the hardware advance and SAP's knowledge of enterprise applications. It enables running OLTP, OLAP, and text processing in a single run-time environment in a scalable way. The foundation of SAP HANA platform is a massively parallel distributed integrated in-memory row and column database system. This talk presents a new paradigm of thinking and architecture underlying SAP HANA platform.

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, p. 2, 2012.
© Springer-Verlag Berlin Heidelberg 2012

Improving the Accuracy of Histograms for Geographic Data Objects

Hai Thanh Mai, Jaeho Kim, and Myoung Ho Kim

Department of Computer Science, KAIST
291 Daehak-ro, Yuseong-Gu, Daejeon 305-701, Republic of Korea
{mhthanh,jaeho,mhkim}@dbserver.kaist.ac.kr

Abstract. Histograms have been widely used for estimating selectivity in query optimization. In this paper, we propose a new technique to improve the accuracy of histograms for two-dimensional geographic data objects that are used in many real-world applications. Typically, a histogram consists of a collection of rectangular regions, called buckets. The main idea of our technique is to use a straight line to convert each rectangular bucket to a new one with two separating regions. The converted buckets, called *bichromatic buckets*, can approximate the distribution of data objects better while preserving the simplicity of originally rectangular ones. To construct bichromatic buckets, we propose an algorithm to find good separating lines. We also describe how to apply the proposed technique to existing histogram construction methods to improve the accuracy of the constructed histograms. Results from extensive experiments using real-life data sets demonstrate that our technique improves the accuracy of the histograms by 2 times on average.

Keywords: databases, query optimization, histogram, selectivity estimation.

1 Introduction

In databases, estimating the selectivities of queries is an essential part of query optimization. Accurate selectivity estimates can help the query execution engine to choose the most efficient query plan. Therefore, over the last decades, the problem of selectivity estimation has been intensively investigated. Several selectivity estimation approaches have been proposed, such as histograms [16,18,4,5,7,21,9,10,20,8,19], wavelet transformation [15,22], singular value decomposition [18], discrete cosine transform [13], kernel estimators [6,10], and sampling [14,11]. Among these approaches, histograms have been shown to be one of the most popular and effective ways to obtain accurate estimates of selectivity [10,8].

Let D be a data set of our interest and S be the data space of D. A histogram H for D consists of a set of m buckets B_i $(1 \leq i \leq m)$ where m is usually a system parameter. Each bucket B_i has a data space S_i that is a subspace of S and a frequency F_i that is the number of data objects in S_i. The data space S_i is

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 3–17, 2012.
© Springer-Verlag Berlin Heidelberg 2012

an interval, a rectangle, or a hyper-rectangle if the data objects have one, two, or higher than two dimensions, respectively. With these m buckets, H approximates the distribution of the data in D. Now, suppose that a query Q on D is given by the user to retrieve data objects within a range S_Q. An estimate of the selectivity of Q (i.e., the number of data objects in S_Q) by using the histogram H, denoted by $F_Q(H)$, is typically computed as: $F_Q(H) = \sum_{i=1}^{m} ((|S_i \cap S_Q|/|S_i|) \cdot F_i)$, under the *intra-bucket uniform distribution assumption*. Here, $|\ |$ denotes the size of data space and $(S_i \cap S_Q)$ denotes the intersecting area of S_i and S_Q. Note, however, that the details of $F_Q(H)$ may differ, depending on the histogram methods. Though uniform distribution of data inside buckets is important for accurate selectivity estimation, it is well-known that such organization of buckets is computationally intractable [17].

In this work, we study the problem of constructing highly accurate histograms for selectivity estimation. We focus on the histograms for two-dimensional geographic data objects where updates do not frequently occur. Objects in this form are generally used in Geographic Information Systems (GISs). The histogram must be constructed so that its estimated selectivity for the query must be close to the true selectivity of the query as much as possible. However, creating an accurate histogram for multi-dimensional data, including geographic data, is not an easy task. When the region of a query fully covers the region of a bucket B, we can use B's object frequency directly. In contrast, when the region of a query partially overlaps with or is fully contained in the region of B, the problem may arise. In these latter cases, the estimated selectivity value for the overlapping region between the query and B is computed in proportion to the size of this overlapping region. Here, if data objects are distributed uniformly within B, our estimation is close to the real object frequency. Otherwise, we are very likely to obtain wrong results. For instance, let us consider a bucket B and a query Q shown in Fig. 1. The size of the overlapping area between Q and B (i.e., the gray area in the figure) is $1/4$ of the size of B. If uniform distribution of objects is assumed, the estimated selectivity of this overlapping region is $1/4$ of the object frequency of B, i.e., 10. Nevertheless, since objects in B are not uniformly distributed and most of them lie at the lower-left part of the bucket whose region does not overlap with Q, the estimate 10 is far from the correct number 1.

In real-life data sets, as the uniformity is rare and non-uniformity is naturally popular, many histogram construction methods have addressed the skewness (i.e., non-uniform distribution) problem of the data, e.g., MinSkew [5], GenHist [10], RkHist [8], STHist [19]. These methods differ from each other in the ways they allocate rectangular buckets onto the data space, so that the data distribution in each bucket is close to uniformity as much as possible. Nevertheless, we have observed that there are many regions, such as the region illustrated in Fig. 1, where it is very difficult to improve the uniformity of data distribution in the bucket further. The reason is that the bucket is a rectangle while the data distribution may have many different shapes. One straightforward solution is to allocate many more buckets to such complex regions. Nevertheless, allocating more buckets to one region means that fewer buckets can be used for other

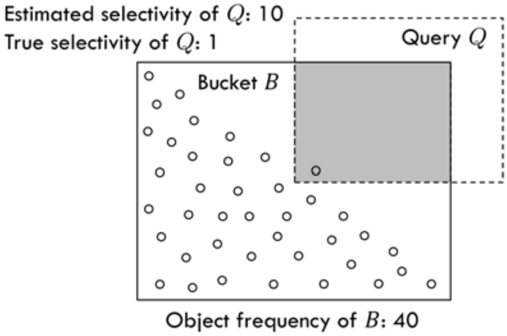

Estimated selectivity of Q: 10
True selectivity of Q: 1
Query Q
Bucket B
Object frequency of B: 40

Fig. 1. Inaccuracy of the histogram when data in the bucket is not uniformly distributed

regions because the bucket quota is limited. Another solution is to use generally polygonal shapes for the buckets instead of rectangles. Polygons can fit the distribution of the data objects better. However, since a much higher amount of memory must be used to describe the polygons than the rectangles in general, much fewer buckets can be used. Moreover, this solution incurs higher complexity than the traditional rectangle-based solution in partitioning the data space and deciding the specific shapes of the polygons.

In this paper, we propose a new technique to improve the accuracy of the histograms. Our main idea is to use a straight line to convert each rectangular bucket to a new one with two separating regions. The converted buckets, called *bichromatic buckets*, can approximate the objects' distribution better while preserving the simplicity of the originally rectangular ones. For converting original buckets to bichromatic buckets, we propose an algorithm to find good separating lines. Then, we present how to apply the proposed technique to existing histogram construction methods. We conducted extensive experiments using real-life data sets. The results demonstrate that our technique can elevate the accuracy of the histograms by more than 4 times in several cases and by 2 times on average.

The remainder of this paper is organized as follows. In Section 2, we sketch the main idea of the proposed technique. In Section 3, we present the proposed technique in detail in the form of an algorithm. We describe how selectivity estimation is done with the new technique in Section 4 and show how to apply this technique to existing histogram methods in Section 5. Experimental results are presented in Section 6, followed by a review of related work in Section 7. Finally, we conclude the paper in Section 8.

2 Sketch of the Proposed Technique

Let us consider Fig. 1 again. As we have mentioned in the previous section, using polygons instead of rectangles can approximate the data distribution better, but incurs much higher memory requirement. In contrast, the rectangles require only small amounts of memory (i.e., for each rectangle, we only need to store the coordinates of two diagonal corner points), while fail to approximate non-rectangular shapes of data distribution. Therefore, we take a hybrid approach as follows. For each rectangular bucket B, we add a straight line to separate B into two disjoint parts so that the object distribution in each part is as uniform as possible. We call this line the *separating line* of B and we call these two parts P_α and P_β. A rectangular bucket equipped with a separating line is called a *bichromatic bucket*. In terms of object frequency, we store the total number of objects in the whole data space of B and the ratio of these objects that lie inside one of the two parts. For simplicity, let P_α be the part that we store the object ratio. In a Cartesian plane, a straight line can be described by the linear equation $y = ax + b$ where a is the slope parameter, b is the intercept parameter, and x, y are the variables. Thus, without loss of generality, we define P_α as the part of B that contains all objects o_i(s) with coordinates (x_i, y_i)(s) where $y_i \geq ax_i + b$, while P_β is the part of B that contains all remaining objects.

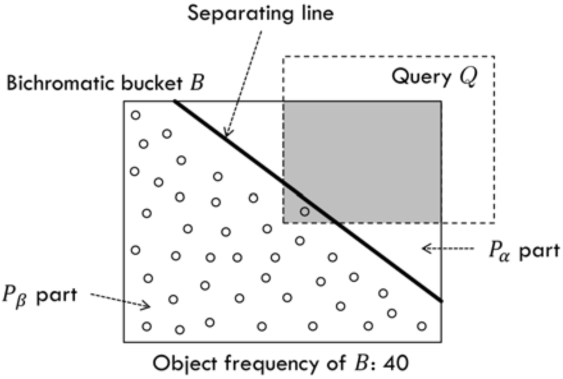
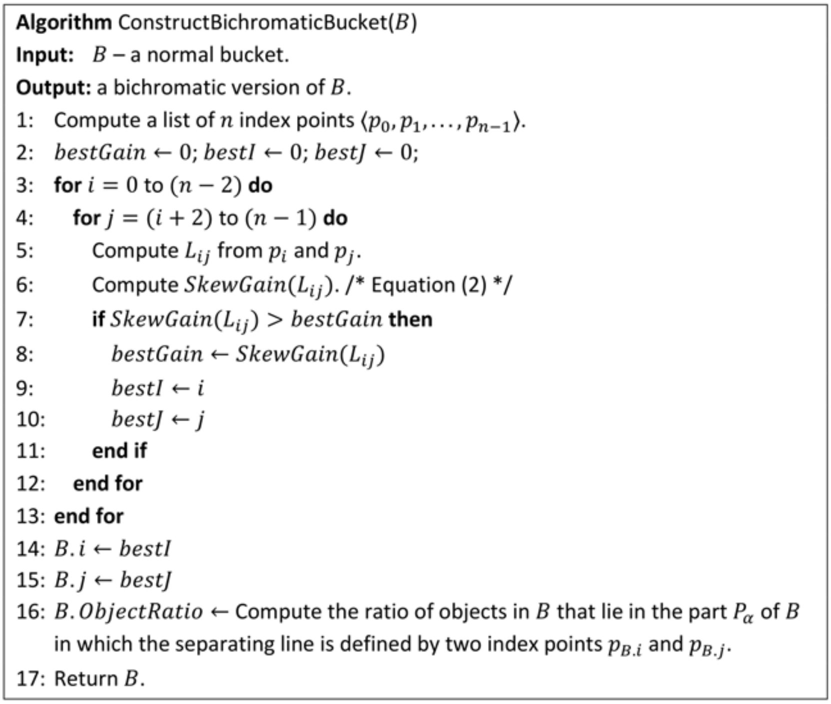


Fig. 2. A bichromatic bucket where data distribution is better approximated only by adding a simple straight line

Let us see Fig. 2 for an example, where data distribution is the same as the one in Fig. 1. It is clear that, the data distribution in any of the P_α and P_β parts of bucket B is more uniform than the data distribution in the whole bucket B without the separating line. Suppose that a query Q is given as in the figure. The estimate of Q's selectivity is computed as the sum of two estimates, i.e., the estimate for the overlapping region between Q and P_α and the estimate for the overlapping region between Q and P_β. This new selectivity estimate is potentially much more accurate than the old estimate which depends on the original version of B. Here, the use of a separating line tends to be very useful as we have observed that, in real-life geographic data, the distribution of objects in a small region may often be divided into two parts by a straight line naturally due to the existence of natural separating factors, such as rivers, mountains, parks, and boundaries between geographic regions.

3 A Bichromatic Bucket Construction Algorithm



```
**Algorithm** ConstructBichromaticBucket($B$)
**Input:** $B$ – a normal bucket.
**Output:** a bichromatic version of $B$.
Compute a list of $n$ index points $\langle p_0, p_1, \ldots, p_{n-1} \rangle$.
$bestGain \leftarrow 0$; $bestI \leftarrow 0$; $bestJ \leftarrow 0$;
**for** $i = 0$ to $(n-2)$ **do**
  **for** $j = (i+2)$ to $(n-1)$ **do**
    Compute $L_{ij}$ from $p_i$ and $p_j$.
    Compute $SkewGain(L_{ij})$. /* Equation (2) */
    **if** $SkewGain(L_{ij}) > bestGain$ **then**
      $bestGain \leftarrow SkewGain(L_{ij})$
      $bestI \leftarrow i$
      $bestJ \leftarrow j$
    **end if**
  **end for**
**end for**
$B.i \leftarrow bestI$
$B.j \leftarrow bestJ$
$B.ObjectRatio \leftarrow$ Compute the ratio of objects in $B$ that lie in the part $P_\alpha$ of $B$
  in which the separating line is defined by two index points $p_{B.i}$ and $p_{B.j}$.
Return $B$.
```

Fig. 3. An algorithm to construct a bichromatic bucket

Algorithm *ConstructBichromaticBucket* in Fig. 3 presents the process of constructing a bichromatic bucket. The input is a normal rectangular bucket B. The output is the bichromatic version of B. In constructing the bichromatic version of B, the most important work is to find a separating line to divide B into two disjoint parts so that the distribution of data objects in each part is as close to uniformity as possible. To do so, in the border lines of B, we define a set of n points, called *index points*, where n is a user-specified parameter and the same value of n is used for constructing all bichromatic buckets in the histogram. The index points are the points that lie on the border lines of B and divide these border lines into equal-length segments. The number of index points on the border line of each side (among 4 sides namely East, West, South, and North), including both two points at the two ends, is $n/4 + 1$. Instead of scanning all possible positions to find the best separating line, we propose to consider only the positions determined by the combinations of index points. Fig. 4a illustrates the index points and a potential separating line, created by connecting an index point to another.

Algorithm *ConstructBichromaticBucket* proceeds as follows.

- We first compute the positions of n index points and store these points in a list $P = \langle p_0, p_1, \ldots, p_{n-1} \rangle$ (Line 1).
- For each point p_i where i is from 0 to $(n-2)$, we examine all points p_j where j is from $(i+2)$ to $(n-1)$ (Line 2 to 13). For each pair i and j, the combination of two points p_i and p_j defines a potential separating line. Let L_{ij} denote this line. We compute L_{ij} (Line 5). Fig. 4b shows all potential separating lines defined by p_0 and other index points. Fig. 4c shows all potential separating lines defined by p_1 and other index points. The dotted lines with increasing color intensity are those potential separating lines. In general, we will consider all these separating lines, starting from the ones defined by p_0 and other index points, then the ones defined by p_1 and other index points, and so on. Among the potential lines started from a point p_i (e.g., p_0 in Fig. 4b and p_1 in Fig. 4c), we illustrate the order of examination by the lines' increasing color intensity.
- Consider a specific line L_{ij}. This line divides the data space of B into two disjoint regions, which we called P_α and P_β. We compute the skewness of the distribution of data objects inside each of these two regions. The skewness of a data region is defined as follows. Note that, this definition is the same as the one in [19].

  Definition 1 (Skewness of a data region). *Consider a data region D. $Skew(D)$, which denotes the skewness of the data distribution in D, is computed as*

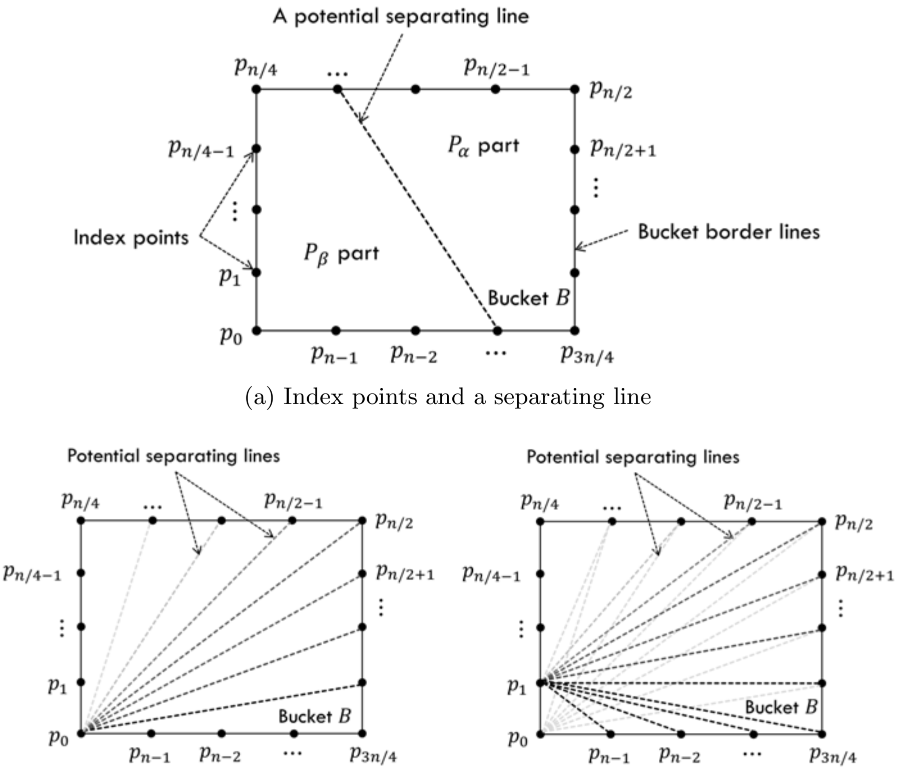
  $$Skew(D) = \sum_r (x_r - \bar{x})^2 \tag{1}$$

  where x_r is the real object frequency at location r and $\bar{x}$ is the estimate of the object frequency based on the uniform distribution assumption within D. In other words, $Skew(D)$ is computed as the sum of squares of absolute errors for all the locations within D.

  For a bucket B, we will use $Skew(B)$ to denote "skewness of the data region of bucket B" if there is no ambiguity. Let $Skew_{ij}(P_\alpha)$ and $Skew_{ij}(P_\beta)$ denote the skewness of the two regions P_α and P_β of bucket B, created by the line L_{ij}, respectively. We compute $Skew_{ij}(P_\alpha)$ and $Skew_{ij}(P_\beta)$. Then, we compute the *potential skewness gain* of using L_{ij} (Line 6) as

  $$SkewGain(L_{ij}) = Skew(B) - (Skew_{ij}(P_\alpha) + Skew_{ij}(P_\beta)) \tag{2}$$

- While we examine potential separating lines L_{ij} ($i = 0..(n-2)$ and $j = (i+2)..(n-1)$), the index i and j of the line L_{ij} that has the highest potential skewness gain is kept track (Line 7 to 11). When the double *for*-loop finishes, the best separating line will be found together with its two index values, *bestI* and *bestJ*. In bucket B, we now store these two values (Line 14 and 15). Note that, L_{ij} can be reconstructed easily if the index values i and j are known. Finally, we compute the ratio between the object frequency of the part P_α and the object frequency of B, and store this ratio in B (Line 16).



(a) Index points and a separating line

(b) Potential separating lines from p_0

(c) Potential separating lines from p_1

Fig. 4. Separating line detection

Running Time Complexity. Let N be the total number of objects in a bucket B and n be the number of index points. Note that, n is a user-specified parameter and it is sufficient to set n at small values (e.g., in the experiments we set $n = 32$). First, computing the locations of n index points takes $\mathrm{O}(n)$ time. Next, we try at most $n^2/2$ potential separating lines. In each trial, computing the separating line needs only a small constant time, but $\mathrm{O}(N)$ time is needed to compute the potential skewness gain. Finally, after the best separating line is found, $\mathrm{O}(N)$ time is used to compute the object ratio in one of the two separating parts. Therefore, the overall time complexity of Algorithm *ConstructBichromaticBucket* is $\mathrm{O}(n + N(n^2/2 + 1))$.

4 Estimating the Selectivity Using a Bichromatic Bucket

Consider an existing histogram construction method M. There is always a procedure that goes together with M to estimate the selectivity for a given query Q. For example, in many methods, the selectivity estimate of Q is computed as the sum of selectivity estimates of Q from all buckets in the histogram. Now, suppose that we apply the proposed technique to M, i.e., to use bichromatic buckets. Then, the general process to estimate the selectivity of Q, using the histogram constructed by M, is not changed. The only difference is in the specific way we use a bichromatic bucket to estimate the selectivity of Q.

Given a bucket B and a query Q. Let $Est(Q, B)$ denote the selectivity estimate of Q using bucket B. If B was a normal rectangular bucket, $Est(Q, B)$ would be computed as

$$Est(Q, B) = \frac{S_B \cap S_Q}{S_B} \cdot F_B \tag{3}$$

where S_Q, S_B, and F_B are the data space of Q, the data space of B, and the object frequency of B, respectively.

However, B is a bichromatic bucket with two disjoint parts P_α and P_β. We compute the selectivity of Q by considering P_α and P_β as two different buckets. For each of these two virtual buckets, the selectivity of Q is computed in the same way as in Equation (3). More specifically, let F_α and F_β denote the object frequencies of P_α and P_β, respectively. Let S_α and S_β denote the data spaces of P_α and P_β, respectively. Since the object frequency of B and the ratio of objects that lie in P_α are known, we can compute F_α and F_β. We can also compute S_α and S_β from the position of B and the values of the index points p_i and p_j that define the separating line. Then, the selectivity of Q is computed as

$$Est(Q, B_{bichromatic}) = Est(Q, P_\alpha) + Est(Q, P_\beta) \tag{4}$$

where

$$Est(Q, P_\alpha) = \frac{S_\alpha \cap S_Q}{S_\alpha} \cdot F_\alpha \tag{5}$$

and

$$Est(Q, P_\beta) = \frac{S_\beta \cap S_Q}{S_\beta} \cdot F_\beta \tag{6}$$

5 Application to Existing Histogram Methods

We show how the proposed technique can be applied to the two existing representative histogram construction methods.

5.1 The MinSkew Method

MinSkew is a well-known histogram construction method for spatial data [5]. Initially, MinSkew approximates the original data set using a uniform grid. Then, it starts with a single bucket consisting of all data objects. For each bucket, it computes the spatial skew of the bucket and the split point along its dimensions that will produce the maximum reduction in spatial skew. Next, MinSkew picks the bucket whose split will lead to the greatest reduction in spatial skew, splits this bucket into two child buckets, and assigns data from the old bucket into the new buckets. After MinSkew finishes, the constructed histogram is a set of non-overlapping buckets. Here, we can simply use the Algorithm *ConstructBichromaticBucket* to get a new version of every constructed bucket. The set of bichromatic buckets is then reported as the final histogram. This strategy, i.e., converting every bucket to bichromatic after the histogram construction method finishes, can be applied to any method where the constructed buckets do not overlap with each other.

5.2 The STHist Method

STHist is a histogram construction method for two or three dimensional geographic data [19]. Given a data set together with the data space, STHist first partitions the entire data space into a number of data segments. Then, for each segment, STHist recursively detects hotspots, which are turned into histogram buckets. Here, a hotspot is a data region that satisfies certain conditions on the object frequency, the shape, and the size. All buckets detected in a data segment are organized into a bucket tree. The histogram is a collection of all these bucket trees. Regarding the bichromatic bucket technique, we can use Algorithm *ConstructBichromaticBucket* to convert every bucket constructed by STHist during the histogram construction process to bichromatic version. In other words, for each data segment, after the root bucket is created, Algorithm *ConstructBichromaticBucket* is applied to this root bucket. Then, STHist detects hotspots inside the improved root bucket and converts these hotspots into child buckets. For each of these child buckets, we apply Algorithm *ConstructBichromaticBucket* to get improved versions. This process continues until no more new bucket is constructed.

6 Performance Evaluation

We compare several existing histogram construction methods, including MinSkew [5], RkHist [8], and STHist [19], with their *bichromatic buckets based* versions, called Bi-MinSkew, Bi-RkHist, and Bi-STHist, respectively.

Data Sets. We use the following 12 real-life data sets for the experiments: i) The *Sequoia* data set [3] that contains 62,556 locations in California; ii) The *Digital Chart of the World* (shortly, *DCW*) data set [3] that contains 19,499 populated places in the United States of America plus Mexico; iii) The *North East* data set [3] that contains 123,593 postal addresses in New York, Philadelphia, and Boston; iv) The *Greece Cities* data set [3] that contains 5,922 cities and villages in Greece; v) The *Cities5000* data set [2] that contains 16,731 cities around the world with a population greater than 5,000; vi) The *Portland Crime* data set [1] that contains 11,846 locations of crime incidents reported to the City of Portland Police Bureau in 2010; vii) The *South Korea* data set [2] that contains 69,000 populated places in South Korea; viii) The *China* data set [2] that contains 64,252 populated places in China; ix) The *Italy* data set [2] that contains 10,881 populated places in Italy; x) The *France* data set [2] that contains 42,834 populated places in France; xi) The *Vietnam* data set [2] that contains 8,624 populated places in

Vietnam; xii) The *Netherlands* data set [2] that contains 4,077 populated places in Netherlands.

Performance Metric. We use the *average relative error* as a performance metric as in [5,10,19]. Average relative error is commonly used to evaluate the accuracy of selectivity estimation. Improving the accuracy is equivalent to reducing the error. Given a query Q, let σ be the actual object frequency of Q, and let σ' be the estimated object frequency of Q by a histogram. Then, the relative error ϵ_{rel} of Q is defined as

$$\epsilon_{rel} = |\sigma - \sigma'|/max\{1, \sigma\} \tag{7}$$

For a set of k queries $\{Q_1, Q_2, \ldots, Q_k\}$, the average relative error E_{rel} is

$$E_{rel} = \frac{1}{k} \cdot \sum_{i=1}^{k} \epsilon_{rel}^{i} \tag{8}$$

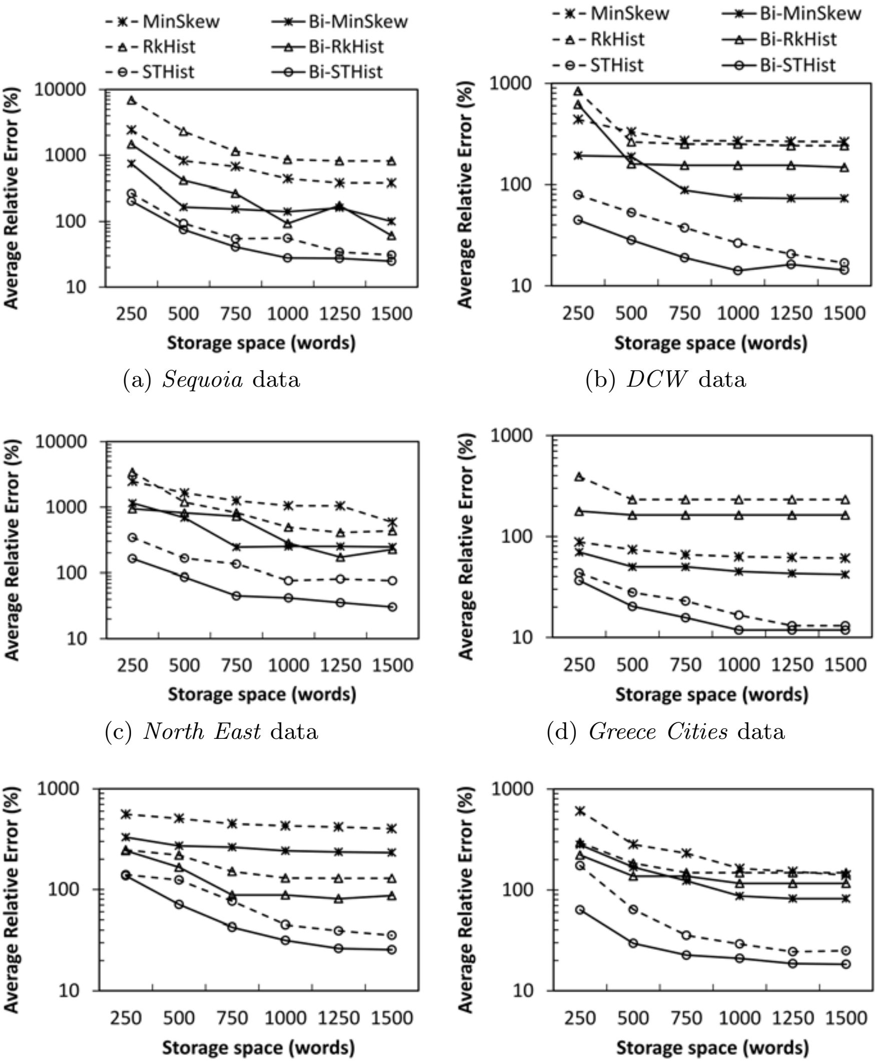
where ϵ_{rel}^{i} is the relative error of query Q_i.

Query Set. We used 100,000 random test queries (i.e., k = 100,000) and computed their average for each point in the result graphs. The locations of the queries are randomly chosen and the sizes of the queries are randomly generated between 0% and 20% of the entire data region.

Bucket Quota Adjustment. Let m be the number of buckets that can be constructed for a histogram of normal buckets. Let m' be the corresponding number of buckets that can be constructed for a histogram of bichromatic buckets. When the proposed technique is used, it is clear that a bichromatic bucket uses more memory than a normal bucket. Thus, m' must be less than m for a fair comparison. Here, for a normal rectangular bucket B, we need to store the coordinates (x_1, y_1) and (x_2, y_2) of two diagonal corner points, and the object frequency of B. Suppose that 4 bytes are generally used to store each coordinate value. Also suppose that 4 bytes are needed to store the object frequency of B. As a result, 5×4 bytes are used for a normal rectangular bucket B. Now, consider the bichromatic version of B, called B'. To store the separating line, we only need to store two index values i and j. We tested different values for the number n of index points and see that $n = 32$ is enough. Thus in our experiments, we set $n = 32$ and only 2 bytes are needed to store both i and j. For the object ratio, 2 bytes is enough for an approximation of high precision. Consequently, B' needs 6×4 bytes while B needs 5×4 bytes for storage. Therefore, given a specific value for m, we will set $m' = RoundDown((5/6) \times m)$ for a fair comparison.

Results. Fig. 5 and 6 show the performance of the histogram methods[1]. The amount of storage space allocated for a histogram is varied from 250 to 1500 words, where the size of each word is 4 bytes. These amounts correspond to 50, 100, 150, 200, 250, and 300 buckets to be constructed for a normal histogram and

[1] Note that, due to space limitation in a page, we separate the set of result graphs into 2 parts and present them in 2 figures (i.e., Fig. 5 and 6) for clarity.



(a) *Sequoia* data

(b) *DCW* data

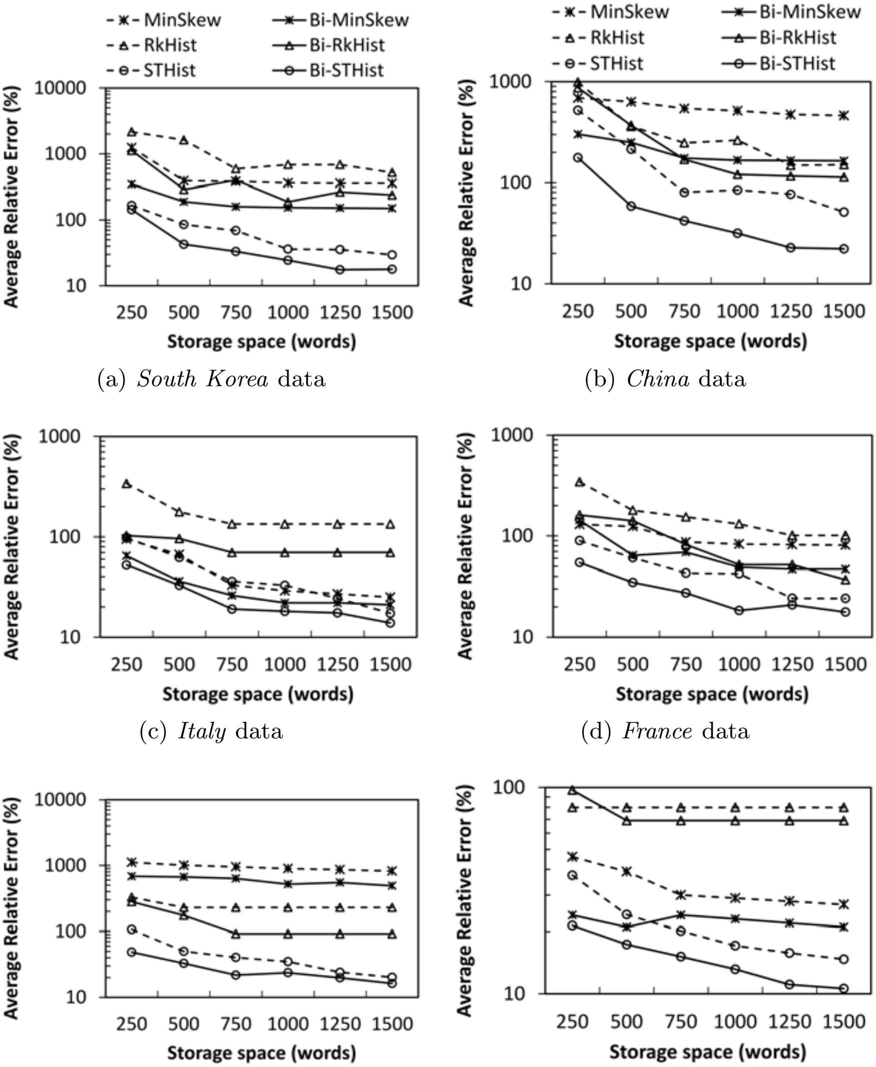
(c) *North East* data

(d) *Greece Cities* data

(e) *Cities5000* data

(f) *Portland Crime* data

Fig. 5. Average relative errors for varying the amount of storage space allocated to a histogram (Part 1 with *Sequoia, DCW, North East, Greece Cities, Cities5000,* and *Portland Crime* data sets)



(a) *South Korea* data

(b) *China* data

(c) *Italy* data

(d) *France* data

(e) *Vietnam* data

(f) *Netherlands* data

Fig. 6. Average relative errors for varying the amount of storage space allocated to a histogram (Part 2 with *South Korea, China, Italy, France, Vietnam,* and *Netherlands* data sets)

to 41, 83, 125, 166, 208, and 250 buckets to be constructed for a histogram with bichromatic buckets, respectively. Note that the Y-axis is shown on a log scale. In general, the average relative errors tend to decrease in all the methods with the increasing amount of storage space (thus, the increasing number of buckets in a histogram). It is because when the number of buckets rises, more accurate statistics can be obtained. In most experiments, the accuracy of the original methods has been significantly improved when our technique is applied. For example, when 1000 words are allocated for the storage space (i.e., 200 buckets are constructed for a normal histogram and 166 buckets are constructed for a histogram of bichromatic buckets) and the *Sequoia* data set is used as in Fig. 5a, the accuracy of Bi-MinSkew (i.e., the MinSkew method with bichromatic buckets) is 3.1 times better than MinSkew (i.e., the original MinSkew method), and the accuracy of Bi-RkHist is 9.2 times better than RkHist. When 1000 words of storage space are used and the *North East* data set is examined as in Fig. 5c, the ratios of accuracy improvement of Bi-MinSkew, Bi-RkHist, and Bi-STHist over the corresponding methods are 4.1, 1.7, and 1.8 times, respectively. This is because in each bichromatic bucket, the whole data space is divided into two separating regions with more uniform data distribution in each region.

7 Related Work

Histograms have a rather long research history. A relatively full record of their history can be found in [12]. In the following, we review some important milestones and concentrate on the studies that are related to ours. Beside the MinSkew [5] and STHist [19] methods which have been reviewed in Section 5, there are several other methods. EquiDepth [16] is the first multi-dimensional histogram method. It attempts to partition the data space, one dimension at a time, into a set of non-overlapping buckets. In the constructed histogram, each bucket contains the same number of data objects. In [18], a method named MHIST-2 was proposed, where at each step the most “critical” attribute is chosen for the partitioning of the data space. In a MaxDiff histogram, at each step MHIST-2 finds the attribute with the largest difference in source values (e.g., spread, frequency, or area) between adjacent values and places a bucket boundary between those values. Thus, when frequency is used as a source parameter, the resulting MaxDiff histogram approximately minimizes the variance of value frequencies within each bucket.

In [10], the authors proposed a histogram method named GenHist. The main difference between GenHist and the previously proposed methods is that GenHist allows buckets to overlap. This new method exploits the fact that, with the same number of bucket quota, the overlapping between buckets permits the data space to be partitioned into a higher number of regions. To build the histogram, GenHist uses multi-dimensional grids of various sizes. High-frequency grid cells are converted into buckets. More recently, a method named RkHist was introduced in [8]. RkHist builds histograms based on an R-tree space partitioning.

It exploits the Hilbert space filling curve to generate an initial space partitioning, then uses a sliding window method, coupled with a new uniformity measure, to further improve the quality of the selectivity estimates.

In addition to statically computed histograms, such as those created by MinSkew [5], GenHist [10], RkHist [8], and STHist [19], there are dynamically generated histograms. For example, the Self-Tuning histogram [4] incrementally maintains buckets in response to feedback from the query execution engine about the actual selectivity of range selection operators. This approach can gracefully adapt buckets to the updates of the underlying data set. Nevertheless, the Self-Tuning histograms are not very good when the data skewness is high [4]. Moreover, in this kind of histograms, because only the regions of queries that have been processed are used, only buckets related to those queries can be updated. Note that, we focus on statically computed histograms in this paper.

Beside histograms, there are alternative methods for selectivity estimation, such as wavelet transformation [15,22], singular value decomposition [18], discrete cosine transform [13], kernel estimators [6,10], and sampling [14,11]. However, histograms have remained the most popular target for selectivity estimation due to their effectiveness and robustness across a wide variety of application domains [12,10,8].

8 Conclusion

Histograms have been widely used for selectivity estimation and constructing highly accurate histograms is an important problem. In this paper, we present a new technique that can be applied to existing histogram construction methods to enhance the accuracy of the constructed histograms. This technique is particularly designed for geographic data objects of two dimensions. The basic idea is to add a straight line to each bucket to divide the bucket into two disjoint parts. In this way, the advantages of traditionally rectangular buckets are preserved while additionally new advantages of polygonal buckets are incorporated. After presenting the motivation, we introduce a simple yet effective algorithm to convert normal rectangular buckets to bichromatic ones. We also described how to apply the proposed technique to existing histogram construction methods. Through extensive experiments using real-life data sets, we show that the proposed technique really improves the accuracy of existing histogram methods. The rate of improvement is about 2 times on average and more than 4 times in several cases.

Acknowledgments. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2011-0020415).

References

1. Crime incidents in 2010 by City of Portland police bureau (2011), http://www.civicapps.org/datasets/crime-incidents-2010
2. The geonames database (2011), http://www.geonames.org
3. R-tree portal (2011), http://www.rtreeportal.org
4. Aboulnaga, A., Chaudhuri, S.: Self-tuning histograms: Building histograms without looking at data. In: SIGMOD Conference, pp. 181–192 (1999)
5. Acharya, S., Poosala, V., Ramaswamy, S.: Selectivity estimation in spatial databases. In: SIGMOD Conference, pp. 13–24 (1999)
6. Blohsfeld, B., Korus, D., Seeger, B.: A comparison of selectivity estimators for range queries on metric attributes. In: SIGMOD Conference, pp. 239–250 (1999)
7. Bruno, N., Chaudhuri, S., Gravano, L.: Stholes: A multidimensional workload-aware histogram. In: SIGMOD Conference, pp. 211–222 (2001)
8. Eavis, T., Lopez, A.: Rk-hist: an r-tree based histogram for multi-dimensional selectivity estimation. In: CIKM, pp. 475–484 (2007)
9. Guha, S., Shim, K., Woo, J.: Rehist: Relative error histogram construction algorithms. In: VLDB, pp. 300–311 (2004)
10. Gunopulos, D., Kollios, G., Tsotras, V.J., Domeniconi, C.: Selectivity estimators for multidimensional range queries over real attributes. VLDB Journal 14(2), 137–154 (2005)
11. Haas, P.J., Swami, A.N.: Sequential sampling procedures for query size estimation. In: SIGMOD Conference, pp. 341–350 (1992)
12. Ioannidis, Y.E.: The history of histograms (abridged). In: VLDB, pp. 19–30 (2003)
13. Lee, J.H., Kim, D.H., Chung, C.W.: Multi-dimensional selectivity estimation using compressed histogram information. In: SIGMOD Conference, pp. 205–214 (1999)
14. Lipton, R.J., Naughton, J.F., Schneider, D.A.: Practical selectivity estimation through adaptive sampling. In: SIGMOD Conference, pp. 1–11 (1990)
15. Matias, Y., Vitter, J.S., Wang, M.: Wavelet-based histograms for selectivity estimation. In: SIGMOD Conference, pp. 448–459 (1998)
16. Muralikrishna, M., DeWitt, D.J.: Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In: SIGMOD Conference, pp. 28–36 (1988)
17. Muthukrishnan, S., Poosala, V., Suel, T.: On Rectangular Partitionings in Two Dimensions: Algorithms, Complexity, and Applications. In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 236–256. Springer, Heidelberg (1998)
18. Poosala, V., Ioannidis, Y.E.: Selectivity estimation without the attribute value independence assumption. In: VLDB, pp. 486–495 (1997)
19. Roh, Y.J., Kim, J.H., Chung, Y.D., Son, J.H., Kim, M.H.: Hierarchically organized skew-tolerant histograms for geographic data objects. In: SIGMOD Conference, pp. 627–638 (2010)
20. Srivastava, U., Haas, P.J., Markl, V., Kutsch, M., Tran, T.M.: Isomer: Consistent histogram construction using query feedback. In: ICDE, pp. 39–51 (2006)
21. Thaper, N., Guha, S., Indyk, P., Koudas, N.: Dynamic multidimensional histograms. In: SIGMOD Conference, pp. 428–439 (2002)
22. Vitter, J.S., Wang, M., Iyer, B.R.: Data cube approximation and histograms via wavelets. In: CIKM, pp. 96–104 (1998)

Improving Online Aggregation Performance for Skewed Data Distribution

Yuxiang Wang, Junzhou Luo, Aibo Song, Jiahui Jin, and Fang Dong

School of Computer Science and Engineering,
Southeast University, Nanjing, P.R. China
{lsswyx,jluo,absong,jhjin,fdong}@seu.edu.cn

Abstract. *Online aggregation* is a commonly-used technique to response aggregation queries with the refined approximate answers (within an estimated confidence interval) quickly. However, we observe that *low selectivity* and *inappropriate sample proportion* significantly affect the online aggregation performance when the data distribution is skewed. To overcome this problem, we propose a Partition-based Online Aggregation System called POAS. In POAS, the side effect of low selectivity can be reduced by efficient pruning of unneeded data due to the partition and shuffle strategies, and the appropriate sample proportion can be achieved as far as possible by drawing samples (tuples) from relevant partitions with *dynamic sample size*. Moreover, POAS applies some statistical approaches to calculate estimates from relevant partitions. We have implemented POAS and conducted an extensive experiments study on the TPC-H benchmark for skewed data distribution. Our results demonstrate the efficiency and effectiveness of POAS.

1 Introduction

In many decision support applications, such as OLAP and data mining tools, aggregation queries are used widely and frequently. The common characteristic of these applications is that large volumes of data need to be accessed and aggregated to response aggregation queries, which is computationally expensive (long time for processing) [1] and resource intensive [2]. However, the exact answers are not always required in real situation, so that the ability to approximately answer aggregation queries efficiently can greatly benefit for these applications.

One commonly-used technique to handle this problem is *online aggregation* [3], which returns approximate answers with their corresponding confidence intervals quickly instead of the precise answers. The basic idea behind this technique is to compute an approximate result against the random samples and refine the result as more samples are received. In this way, users can grasp the overall progress of the running queries and terminate these queries prematurely if an acceptable answer can be arrived at quickly.

However, we observe that online aggregation usually performs poorly when the data distribution is skewed. There are at least two factors that skewed data distribution affects online aggregation: *low selectivity* and *inappropriate sample*

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 18–32, 2012.
© Springer-Verlag Berlin Heidelberg 2012

proportion[1]. For the first factor, the number of relevant tuples which satisfy the query predicate may be much less than others due to the skewed data distribution. Then, there will be few or no relevant tuples in the sample during the initial stage of online aggregation, so that approximate result calculated from such a small set of relevant tuples may lead to large error (need to be continually refined). For the latter factor, the acceptable estimate can be calculated quickly only when the samples are drawn from relevant relation with an appropriate proportion. Unfortunately, this appropriate sample proportion can not be archived with high probability in the initial stage of online aggregation due to the skewed data distribution, leading to large error and extending the processing time.

In this paper, we study the problem of efficiently decrease the effect of skewed data distribution for online aggregation. We propose a Partition-based Online Aggregation System called POAS to overcome the limitations mentioned above. For low selectivity, POAS divides the original dataset into a set of partitions, which are indexed by NRB-T (Nested Red-Black Tree). Each query will run against the relevant partitions, which overlap with the query range, instead of the whole dataset. In this way, the selectivity of aggregate query would increase due to the efficient pruning of unneeded data. On the other hand, POAS draws samples from relevant partitions with *dynamic sample size* to achieves the appropriate sample proportion as far as possible, then the acceptable running estimate to the final result could be refined in the early stage of online aggregation, saving execution time significantly.

We have implemented POAS and compared it with the COSMOS, which is a latest online aggregation system for centralized environment proposed in [1], on the TPC-H benchmark for skewed data distribution. The experiment results show that our POAS can eliminate the effect of skewed data distribution effectively and improve online aggregation performance observably.

The main contributions of this paper are summarized as follows:

1. We point out the limitations of online aggregation for skewed data distribution and present two fundamental principles that need to be considered in the problem solution.
2. We propose a new online aggregation system called POAS based on these two fundamental principles to improve the online aggregation performance for skewed distribution.
3. We implement POAS and conduct extensive experiments, the results demonstrate the efficiency and effectiveness of POAS.

The rest of this paper is organized as follows. In the next section, we give a brief overview of related work. In Section 3, we point out the limitations of online aggregation for skewed data distribution and present two principles to solve it. In section 4, we give an overview of POAS and introduce the major components in details. Section 5 presents the statistical estimators used in POAS. And in section 6, we report results of the experimental evaluation. Finally, we conclude this paper in Section 7.

[1] Sample proportion indicates the proportion of samples in the sample.

2 Related Works

In many real applications, such as OLAP, aggregation queries are used widely and frequently. However, calculate exact results for these queries incurs long response time, and is not always required. To response queries in a short processing time with the acceptable results, approximate query processing (AQP) is proposed recently.

Online aggregation [3] is one commonly-used AQP technique to provide a time-accuracy tradeoff for aggregation queries. Approximate answer within a running confidence interval is produced during early stages of query processing and gradually refined until satisfies the users expectation. The running confidence interval indicates to user that the estimated proximity of each running aggregation query to its final result. In [4], Haas illustrates how the central limit theorems, simple bounding arguments and the delta method can be used to derive formulas for both large-sample and deterministic confidence intervals.

To support join operation for online aggregation, Hass and Hellerstein introduced a novel join methods called ripple joins in [5]. But the convergence of ripple joins can be slow when memory overflows. To handle this problem, hash ripple join algorithm is proposed in [6], which combines parallelism with sampling to speed convergence and also maintains good performance in the presence of memory overflow. However, all works in [3,4,5,6] are focused on single query processing rather than multi query optimization. Therefore, Wu et al. proposed a new online aggregation system called COSMOS to process multiple aggregation queries efficiently [1]. COSMOS organizes queries into a dissemination graph to exploit the dependencies across queries, and the partial answers can be reused by the linked queries.

More recently, some online aggregation systems for distributed context are proposed along with the development of P2P and cloud computing [7,8,9,10]. Wu et.al. extend the online aggregation to a P2P context where sites are maintained in a DHT network [7], which maintains synopses that can be reused by different queries. In addition, [8,9] demonstrates a modified version of Hadoop MapReduce framework that supports online aggregation. And [10] proposed a new online aggregation system that supports MapReduce job based on the open-source project Hyracks [11], which discuss a Bayesian framework for producing estimates and confidence intervals for online aggregation.

However, none of the above papers address the limitations of online aggregation for skewed data distribution that we have discussed in this paper. In particular, the skew issue has been considered in the sample-based AQP technique rather than online aggregation and the outlier-indexing and weighted sampling are proposed to provide an aggregate result with significantly reduced approximation error [2]. But the two techniques do not apply to online aggregation due to the following reasons: (1) they do not satisfy the requirement of online aggregation for unbiased sampling and (2) they are unsuitable for the iterative query processing model of online aggregation.

3 Effect of Skewed Data Distribution

In this section, we take AVG as an example to demonstrate the limitations of online aggregation for skewed data distribution: (1) the low selectivity, and (2) the inappropriate sample proportion.

For the first limitation, the number of tuples drawn from relevant relation will be proportional to its size since the sampling of online aggregation is unbiased when the sample size is "small enough" [4]. Given a query Q with 1000 tuples and the selectivity is 1%. Suppose 50 tuples are sampled each time to refine the estimate iteratively. Then, the event that the sample contains no relevant tuples in the first iteration has probability of 0.60, contains one tuple with probability of 0.31, and 0.076 for two tuples, etc. Therefore, few or no relevant tuples may in the sample during the early stage of online aggregation due to the low selective, so that AVG is calculated from the sample contains such a small set of relevant tuples may lead to large error and extend the processing time.

For the second limitation, there are some tuples contribute a lot to the query (important tuples) in relevant relation. If the sample contains insufficient/superabundant important tuples, which means the proportion of relevant tuples is inappropriate, it may lead to large error and extend the processing time. Consider the same relation R, and the query Q with selectivity of 99%. Let A be part of relevant relation contains 9800 tuples with value 1, while the remaining 100 tuples with value 100 (important tuples), belong to part B. Thus, the average over all relevant tuples of R is 2. Given the error rate 0.01, then the appropriate proportion between A and B ranges from 96:1 to 100:1. Suppose the sample contains 200 relevant tuples in the first iteration. Note that, if three or more tuples from B were to be included in the sample, then the estimate would be more than 2.485 (sample proportion is less than 65:1). On the other hand, if one tuple from B in the sample, the estimate is 1.495 (sample proportion is 199:1). Only in the event where two tuple from B in the sample (sample proportion is 99:1), leading to an acceptable estimate to AVG, which is 1.99, with probability of 0.27. Therefore, the running estimate to AVG need to be gradually refined, which extends the processing time.

Although these two examples demonstrated the limitations for AVG, similar arguments also hold for the sum and count with a little difference due to the different estimators used for estimating.

Based on the analysis given above, we present two basic principles to overcome the limitations of online aggregation for skewed data distribution: (1) Efficient **pruning of unneeded data** to increase the selectivity of aggregate query. (2) Draw samples **with an appropriate sample proportion** to produce a good approximate result as soon as possible.

In the following sections we propose POAS to resolve these skewed issues taking into account these two principles.

4 Partition-Based Online Aggregation System

4.1 System Overview

Figure 1 shows the system architecture of POAS, which comprises three major components: *Data Preprocessor*, *Data Management* and *Query Engine*. Data preprocessor re-organizes the original dataset as partitions. Each query will run against part of the partitions instead of the original whole dataset. In this way, the selectivity of query would increase due to the **efficient pruning of unneeded data** (the proportion of relevant tuples becomes higher than before). Moreover, each partition is shuffled so that sequentially scanning each partition gives rise to the random sampling with lower I/O cost. In data management component, we store each partition as a separate file and index them by NRB-T (Nested Red-black Tree). Through NRB-T, queries can obtain their relevant partitions, which overlap with the search range of queries, efficiently. For query engine component, samples are drawn from these relevant partitions **with dynamic sample size to archive appropriate sample proportion** as far as possible, so that make the acceptable running estimate to the final result could be refined in the early stage of online aggregation.

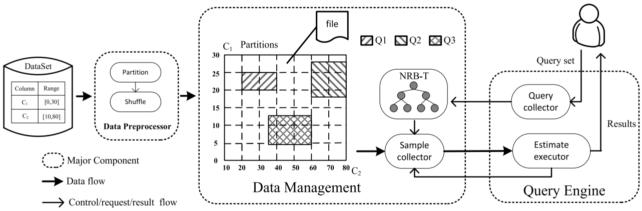


Fig. 1. System Architecture of POAS

4.2 Data Preprocessor

In this subsection, we discuss two aspects of data preprocessor: partition phase and shuffle phase.

Partition Phase. The task of partition phase is to manage the original dataset in the granularity of partitions. Such processing can be implemented quite efficiently by making two scans of the dataset. The purpose of the first scan is to determine two input parameters of partition phase: the value range of columns $\mathcal{R}$ and the partition size of columns $\mathcal{N}$.

Definition 1. $\boldsymbol{\mathcal{R}}$, $\boldsymbol{\mathcal{N}}$. *Given a dataset with n partitioning columns $\mathcal{C} = \{C_1, C_2, ..., C_n\}$, the value range of columns denotes by $\mathcal{R} = \{R_1, R_2, ..., R_n\}$, where R_i represents the value range of C_i. And the partition size of columns denotes by $\mathcal{N} = \{N_1, N_2, ..., N_n\}$, where N_i represents the partition size of C_i.*

In our implementation, $\mathcal{N}$ is predefined by system. If $\mathcal{R}$ is already available in the meta-data, the first scan may be omitted. Based on the two parameters, we can obtain the identities of all partitions, denoted by $\mathcal{P}$. To simplify the presentation, we define $\mathcal{P}$ as follows.

Definition 2. $\boldsymbol{\mathcal{P}}$. *Each column C_i is divided into N_i uniform intervals, denoted by $\mathcal{I}_i = \{I_i^1, I_i^2, ..., I_i^{N_i}\}$. Thus, $\mathcal{P} = \mathcal{I}_1 \times \mathcal{I}_2 \times ... \times \mathcal{I}_n$, where $|\mathcal{P}| = \prod_{i=1}^{n} N_i$. Each partition is identified by a certain $P_i \in \mathcal{P}$.*

Initially, all of these partitions are empty so that the purpose of the second scan is to fill in them with corresponding tuples. After these two scans are completed, the original dataset is re-organized as partitions.

Shuffle Phase. The task of shuffle operation is to gain the randomness of tuples for each partition, so that sequentially scanning the partition can reduce the I/O cost of sampling (completely random disk access can be five orders of magnitude slower than sequential access [12]). The idea of sequentially scanning a dataset have also applied in several other works [13,14,1]. Our POAS shares similar motivation to the scrambled dataset proposed in [1]. The strategy they used is to "random write" the tuples into the scrambled dataset, which gains the great randomness of tuples but cost a lot of time to seek position randomly in the scrambled dataset.

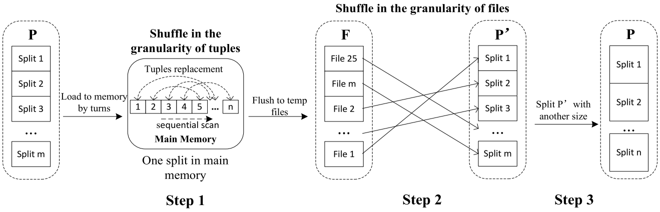


Fig. 2. Shuffle of one partition

In our implementation, a two-level shuffle operation is deployed to avoid the side effect of "random write" since the "random write" is replaced by "sequential write" as shown in Figure 2. Given a partition P, we conduct shuffle operation as follows: step 1, divide P into m splits with equal size s, each split will be loaded into main memory by turns ($s \leq$ the size of main memory) and a tuple-level shuffle is condected to gain the randomness of tuples for each split: as we scan a tuple T, we exchange it with a randomly chosen victim tuple V, each shuffled split in the main memory is **sequential written** to a temporary file with a random numeric file name; step 2, a file-level shuffle will **sequential writes** each $f \in F$ into an intermediate partition P' by order of the file name, which gains the randomness of splits for partition P; step 3, repeat the above processes with a different split-size for k times (to gain a good randomness, we set $k = 4$ and use four split-size which are prime to each other).

4.3 Data Management

The goal of data management is to lookup the partitions that overlap with a given probe search range efficiently by NRB-T, and make POAS much more scalability by incremental update strategy.

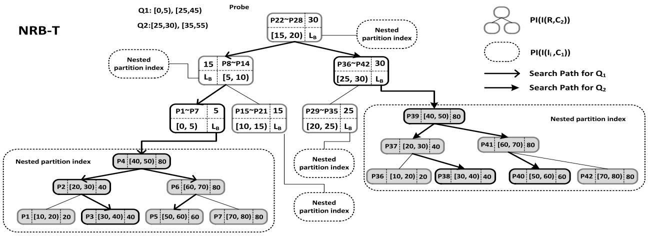


Fig. 3. NRB-T for the dataset shown in Figure 1

Nested Red-Black Trees. The core idea behind NRB-T is to associate each partition with its intervals in a hierarchical model. Then, the NRB-T enables efficient lookup of the relevant partitions for given query. To simplify the presentation, we have some definitions as follows:

Definition 3. $\boldsymbol{I(R, \mathcal{C}_i)}$. *Given a dataset $\mathcal{R}$ with n partitioning columns $\mathcal{C} = \{C_1, C_2, ..., C_n\}$, the $I(\mathcal{R}, C_i)$ denotes a set of intervals of dataset R for C_i.*

Definition 4. $\boldsymbol{PI(I)}$. *Given a set of intervals I based on Definition 5, the $PI(I)$ denotes the partition index for I. Each $PI(I)$ is created by the High values of the intervals.*

Definition 5. $\boldsymbol{BS(\mathcal{C})}$. *Given the partitioning columns $\mathcal{C}$ of $\mathcal{R}$, $BS(\mathcal{C})$ denotes the build sequence of NRB-T. A NRB-T is built in hierarchical model corresponding to a given $BS(\mathcal{C})$.*

We take the dataset mentioned in Figure1 as an example to describe the build phase of NRB-T: step 1, $\mathcal{C}$ is ordered in descending by their predefined partition size, denoted by $\mathcal{C}'$ and we set $BS(\mathcal{C}) = \mathcal{C}'$; step 2, we build the partition index $PI(I(R, C_2))$ as shown in Figure 3, each node comprises four annotations:*Partitions*, *Interval*, *Max* and *Pointer* where the *Max* annotation records the maximum *High* value across both its subtrees; step 3, we build the nested partition index $PI(I(I_i, C_1))$ for $\forall I_i \in I(R, C_2)$.

Moreover, NRB-T also has an acceptable complexity. Take a dataset with two partitioning columns as an example. Suppose N_1 and N_2 ($N_1 \leq N_2$) are the partition size of each column. Then, building NRB-T has $O(N_1 \log N_1 + N_1 \times N_2 \log N_2)$ complexity, and the memory needs are $\theta(N_1 + N_1 \times N_2)$. Probing a NRB-T takes $O(min(N_1, k_1 \times \log N_1) + min(N_1, k_1 \times \log N_1) \times min(N_2, k_2 \times \log N_2))$, where k_1 and k_2 are the numbers of matching intervals for each partition index.

Update Strategy. The update strategy of POAS is designed based on the fact that the updates to the data warehouse system are performed in a batch mode. We collect all the updates and commit them periodically. During the update period, our system will stop processing any queries and only focus on two operations: update to partition and update to NRB-T (if necessary). Given number of tuples need to be inserted, the update strategy is performed as follows:

Case 1: Large Number of Update Tuples. Firstly, we partition and shuffle these tuples to generate a set of update partitions. Then, we append them to the end of the corresponding exist partitions, and conduct another file-level shuffle operation for better randomness. If there are tuples not belong to any partitions, several new partitions are created and extra nodes are added to NRB-T recording these new partitions.
Case 2: Small Number of Update Tuples. We just append the update partitions to the end of corresponding partitions without shuffle operation and implement the same update operation to NRB-T as Case 1 if necessary.

4.4 Query Engine

In this subsection, we show how the aggregation queries are processed in the query engine. Firstly, query engine access NRB-T to obtain relevant partitions for a given query. Then, samples are drawn from these partitions with dynamic sample size. Finally, the estimate to the final result is calculated and refined by applying the formulas introduced in section 4, and the query can be terminated in the following two cases: (a) When the estimate to the final result satisfies the expectation of user; (b) When the query has scanned all tuples and generates the accurate result.

Dynamic Sample Size. The basic idea behind dynamic sample size is each relevant partition provides the number of samples that is proportional to its cardinality. For example, suppose the query Q needs to draw k samples from a set of relevant partitions S_{rp}. Then, there are $\frac{|rp_i|}{|S_{rp}|} \cdot k$ samples are drawn from each $rp_i \in S_{rp}$. In this way, the appropriate sample proportion between these relevant partitions can be archived as far as possible. The reason we call it dynamic is that the sample size of a given partition is different for each query as the relevant partitions of each query are different. Since the requirement of unbiased sampling of online aggregation [3], we need to proof the samples collected with dynamic sample size are unbiased.

Theorem 1. *With dynamic sample size, the samples drawn from S_{rp} are unbiased.*

Proof. The samples for Q is unbiased if each tuple in S_{rp} has the same probability of being picked. Sequential scan to each rp_i results in each tuple has the same probability of being selected, which is approximate as $\frac{1}{|rp_i|}$. In addition, each rp_i has probability of $\frac{|rp_i|}{|S_{rp}|}$ to draw samples due to the dynamic sample size. Then, each tuple in S_{rp} has the same probability $\frac{|rp_i|}{|S_{rp}|} \cdot \frac{1}{|rp_i|} = \frac{1}{|S_{rp}|}$ of being sampled.

5 Estimators and Confidence Intervals for SUM, COUNT and AVG

Let ε', c be the running error bound and confidence respectively, ε' and c give probabilistic estimate of approximate result v' which means exact result v lies in the interval $[v' - \varepsilon', v' + \varepsilon']$ with probability c. In this section, we take SUM, COUNT, AVG as examples to show how estimates and confidence intervals can be obtained.

5.1 Queries for Single Relation

We consider the single relation query firstly:

SELECT ***op(expression)*** **FROM** ***R*** **WHERE** ***predicate***

Given relevant partitions S_{rp} for this query, and a sample set S has been obtained from S_{rp} with dynamic sample size. Then, the natural estimators for SUM, COUNT and AVG are:

$$v'_{s|c} = \frac{|S_{rp}|}{|S|} \cdot \sum_{t_i \in S} expression_p(t_i) \qquad v'_a = \frac{v'_s}{v'_c} \tag{1}$$

where $expression_p\,(t_i)$ equals t_i for SUM and 1 for COUNT if t_i satisfies the predicate, and 0 otherwise.

The estimators for SUM and COUNT are *unbiased* means that $v'_{s|c}$ would be equal on average to the exact query result if the sampling and estimation process were repeated over and over. And they are also *consistent* means that it converges to the exact query result as more and more tuples are sampled. Moreover, the estimator of AVG is consistent due to the consistency of $v'_{s|c}$. Although v'_a is biased, but the bias is typically negligible as the number of sampling steps increases [5].

Based on Central Limit Theorem, $\frac{\sqrt{|S|}\cdot(v'-v)}{\sigma}$ is distributed approximately as a standardized normal distribution when $|S|$ is "large enough", where $\frac{\sigma^2}{|S|}$ is the variance of v'. Given an predefined confidence c, the error bound ε' can be computed by the formula(2) and $P\{|v' - v| \leq \varepsilon'\}$ is the predefined confidence c.

$$P\{|v' - v| \leq \varepsilon'\} \approx 2\Phi\left(\frac{\varepsilon'\sqrt{|S|}}{\sigma}\right) - 1 \tag{2}$$

5.2 Queries for Multi-relations

As we know that the method used in [5] might be less efficient since much more unnecessary join operation between $S_{rp}(R)$ and $S_{rp}(S)$ is processed. Figure 4 shows the partitioning of relation R and S for the following query:

SELECT ***op(expression)*** **FROM** ***R,S*** **WHERE** $R.C_i = S.C_i$ **and** $0 \leq R.C_1 \leq 10$ **and** $0 \leq S.C_2 \leq 15$

The shaded area represents $S_{rp}(R)$ and $S_{rp}(S)$ respectively. Note that, join plan $\bigcup_{i=1}^{4} rp_i(R) \bowtie \bigcup_{i=1}^{2} rp_i(S)$ will produce some unnecessary join pairs as $rp_1(R) \bowtie rp_2(S)$, $rp_3(R) \bowtie rp_2(S)$, $rp_2(R) \bowtie rp_1(S)$ and $rp_4(R) \bowtie rp_1(S)$.

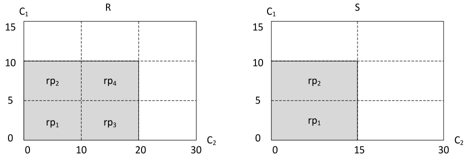


Fig. 4. Partitioning of relations R,S

To further improve the performance of POAS, we prune the unnecessary join pairs by NRB-T firstly. Let s_i be the set of samples from $rp_i(R)$ and s_i' from $rp_i(S)$ with dynamic sample size, where $\sum |s_i| = |S_R|$ and $\sum |s_i'| = |S_S|$. Then, the estimators for SUM and COUNT of each jp_i can be calculated as [5], denote by $v'_{(s|c,i)} = \frac{|jp_i|}{|S_i|} \cdot \sum_{(r,s)\in jp_i} expression_p(r,s)$, where S_i be the set of samples from jp_i, e.g. $|S_1| = |s_1 + s_3| \cdot |s_1'|$. And the estimator for AVG is calculated as $v'_{(a,i)} = \frac{v'_{(s,i)}}{v'_{(c,i)}}$. Then, we can obtain the estimators of multi-relational query by:

$$v'_{s|c} = \sum_{i=1}^{|jp|} v'_{(s|c,i)} \qquad\qquad v'_a = \frac{v'_s}{v'_c} = \frac{\sum_{i=1}^{|jp|} v'_{(a,i)} \cdot |S_i'|}{\sum_{i=1}^{|jp|} |S_i'|} \tag{3}$$

where $|S_i'|$ is the number of samples in S_i that satisfy the predicate. Note that, the estimator $v'_{s|c}$ is unbiased since:
$E(v_s') = \sum_{i=1}^{|jp|} E(v'_{(s,i)}) = SUM(Q)$ and $E(v_c') = \sum_{i=1}^{|jp|} E(v'_{(c,i)}) = COUNT(Q)$. And the bias of estimator v_a' converges to 0 as the number of sampling steps increases. Moreover, these estimators are consistent due to the same reason in section 5.1.

Since $v'_{(s|c|a,i)}$ has approximately a normal distribution with mean $v_{(s|c|a,i)}$ and variance $\frac{\sigma^2_{(s|c|a,i)}}{|S_i|}$ based on the analysis in [5], and the linear combination of multiple independent normally distributed random variables still has a normal distribution, then $v'_{s|c|a}$ has a normal distribution with mean $v_{s|c|a}$ and variance $\sigma^2_{s|c} = \sum_{i=1}^{|jp|} \frac{\sigma^2_{(s|c,i)}}{|S_i|}$ and $\sigma_a^2 = \frac{\sum_{i=1}^{|jp|} \sigma^2_{(a,i)} \cdot |S_i|}{(\sum_{i=1}^{|jp|} |S_i|)^2}$.

According to the basic feature of normal distribution, $\frac{v'-v}{\sigma}$ is distributed as standardized normal random variable. Then, the error bound ε' can be computed by the following formula:

$$P\left\{|v' - v| \le \varepsilon'\right\} \approx 2\Phi\left(\frac{\varepsilon'}{\sigma}\right) - 1 \tag{4}$$

6 Experiments

6.1 Experimental Setup

We have implemented POAS in java and deployed it on a IBM System x3500 with 2 Quad-Core Intel Xeon CPU E5335 and 4GB memory. A modified TPC-H toolkit [15] is employed to generate 100G dataset as our test data with Zipf distribution determined by the Zipf parameter **z** (**z** varies over 0, 1.2, 1.6, where 0 represent the uniform distribution). We generate queries based on the Single Talbe Template (T_1) and Multi-table Template (T_2):

T_1: **SELECT sum(C_i)|count(C_i)|avg(C_i) FROM LINEITEM**
WHERE $[l\_discount > x$ **and** $l\_discount < x + y] \mid [l\_quantity > x$ **and** $l\_quantity < x + y]$
$[l\_extendedprice > x$ **and** $l\_extednedprice < x + y]|$

T_2: **SELECT sum(C_i)|count(C_i)|avg(C_i) FROM LINEITEM L, ORDERS O**
WHERE $L.orderkey = O.orderkey$ &
$[l\_discount > x$ **and** $l\_discount < x + y] \mid [l\_quantity > x$ **and** $l\_quantity < x + y]$
$[l\_extendedprice > x$ **and** $l\_extednedprice < x + y]|$

The aggregate type (AVG, COUNT, SUM) for each query is random selected during the query generation process. To test the effectiveness of POAS for different selectivity, we vary y from 3% to 20% of the value range of C_i for random x. For comparison purposes, we implemented 3 methods. In *POAS*, the queries are processed by our POAS directly, we partition and shuffle the two relations based on the columns in "where" predicates (the default partition size for each partitioning column is 6). In *COSMOS*, we deploy the latest online aggregation system called COSMOS proposed in [1]. In *OA-original*, the queries are processed by the original online aggregation in [3,5].

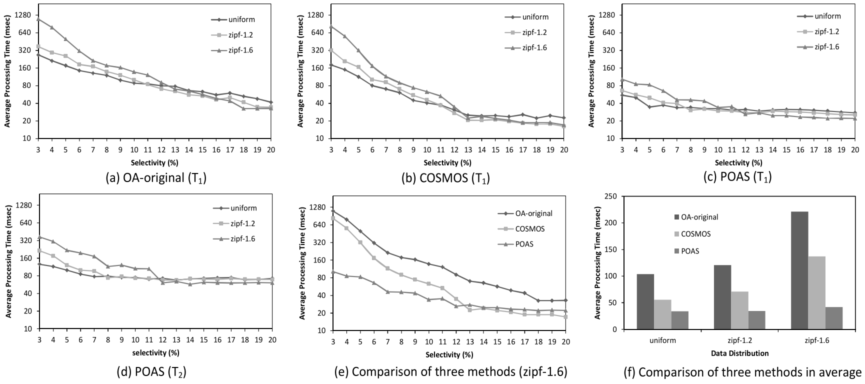
In our experiments, we use the average processing time of the query as the metric. The default error rate e is 0.01 and confidence c is 95%. Each experiment executes 1800 queries (100 queries for one selectivity) and repeated 10 times to remove any side effects.

6.2 Performance Comparison

We evaluate the performance of three online aggregation schemes against 100G test data with different data distribution. Since the results of template 1 and template 2 show similar trend, we present all results of template 1 and part of results of template 2. As shown in Figure 5(a) and (b), *OA-original* and *COSMOS* performs poorly when the selectivity is relatively small, and the skewed data distribution makes such weakness much more obviously. On the other hand, our *POAS* performs stable for different selectivity and the adverse effect of skewed data distribution is eliminated significantly as shown in Figure 5(c). However, the performance improvement of *POAS* slows down when selectivity larger than 15% and close to the performance of *COSMOS* as shown in Figure 5(e). This is because the effect of skewed distribution for higher selectivity is much less than before. Figure 5(f) shows the comparison of average performance for all selectivity (3% $\sim$ 20%), *POAS* outperforms the other methods and efficiently decrease the effect of skewed data distribution.

6.3 Effect of Error Rate and Confidence

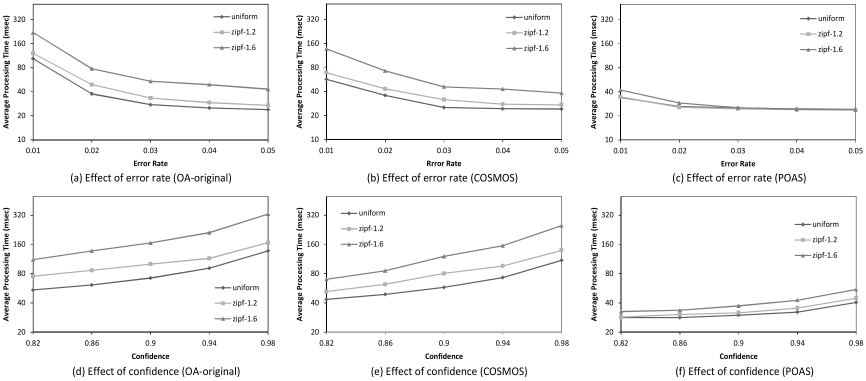
Given a predefined confidence, the smaller error rate indicates the more approximate result we can obtain. And a larger confidence gives the higher probability



(a) OA-original (T_1)
(b) COSMOS (T_1)
(c) POAS (T_1)
(d) POAS (T_2)
(e) Comparison of three methods (zipf-1.6)
(f) Comparison of three methods in average

Fig. 5. Performance of Three Online Aggregation Systems

that the accurate result is bounded by the estimated error bound. In this test, we vary the predefined error rate and confidence respectively to examine the performance of the three methods. The predefined error rate ranges from 0.01 to 0.05.



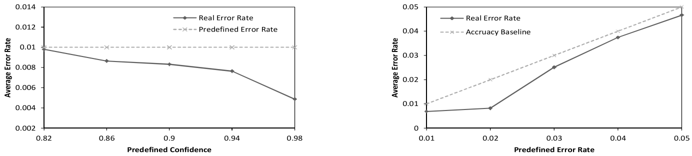
(a) Effect of error rate (OA-original)
(b) Effect of error rate (COSMOS)
(c) Effect of error rate (POAS)
(d) Effect of confidence (OA-original)
(e) Effect of confidence (COSMOS)
(f) Effect of confidence (POAS)

Fig. 6. Effect of Error Rate and Confidence

Figure 6(a-c) shows all methods have the similar trend that much more samples is needed to gain the higher precision, which leads to long processing time. And the skewed data distribution makes such weakness much more obviously for *OA-original* and *COSMOS*. However, the processing time of *POAS* for different data distribution reduces gradually and reaches stable at around 25 msec as shown in Figure 6(c) which means the affect of skewed data distribution is eliminated effectively. On the other hand, we vary predefined confidence from 82% to 98% to show the effect of confidence. Figure 6(d-f) show that along with the increase of confidence, much more samples are received to update the estimators (explained by Formula (2)), which lead to a longer processing time. However, the performance curve of *POAS* is relatively smooth without any drastic change and scalable with skewed data distribution, too.

6.4 Precision of Estimation

In this test, as the results of two templates for different data distribution show similar trend due to the CLT is used in estimation, we only present the result of *POAS* for template 1. We use the average real error rate of queries as the metric to show the effect of error rate and confidence on the accuracy of *POAS* (real error rate is calculated as $\frac{|v-v'|}{v}$, v is computed by PostgreSQL).



(a) Accuracy for different confidence

(b) Accuracy for different error rate

Fig. 7. Accuracy of Estimation

We vary the predefined confidence from 82% to 98% and set the error rate to 0.01 (Zipf 1.6). Based on the result depicted in Figure 7(a), the average real error rate is always lower than the predefined error rate, which means most of queries have gained a good approximate result. In addition, the average real error rate also decreases with the increase of confidence. This is expected as the higher confidence leads to more samples are received to gain a better estimation.

Moreover, we vary the predefined error rate from 0.01 to 0.05 and set the confidence to 95% (Zipf 1.6). As shown in Figure 7(b), the estimation is quite accurate because the average error rate is always lower than the predefined error rate (*real error rate* is under the *baseline*).

6.5 Effect of Partition Size

In this test, we show the effect of partition size on POAS. For facilitate to discuss, each column has the same partition size (PS) which varies from 2 to 10. As shown in Table 1, average processing time reduces with increase of partition size. This is expected as the query can prune much more unneeded data due to the fine-grained partition (larger partition size). But the performance improvement is getting slower with increase of partition size. When we increase partition size from 6 to 10, the decline of average processing time is much slower than the cases with smaller partition size. This is because much more I/O cost is caused by accessing to the larger number of partitions. Therefore, the bigger partition size is not always optimal, we should choose the appropriate partition size by considering the real application environment.

Table 1. Effect of Partition Size (data size=100G e=0.01 c=95%)

Partition Size	Processing Time (msec)				
	PS=2	PS=4	PS=6	PS=8	PS=10
uniform	60.322	39.87	34.47	33.19	32.34
zipf-1.2	65.83	46.29	35.11	34.66	34.11
zipf-1.6	124.56	90.45	43.16	42.54	41.20

6.6 Preprocessing Performance

In this test, we compare two preprocessing methods, *shuffled partition* (deployed in POAS) and *scrambled dataset* (deployed in COSMOS). The performance of preprocessing is affected by data size rather than other factors, as both two methods need to scan the whole dataset regardless of what data distribution or partition size they have. We only present the result with uniform distribution and partition size 6. As shown in Table 2, the *POAS* performs better than *COSMOS* since the "random write" is replaced by "sequential write". And the processing time of *POAS* is approximate proportional to the data size, which is superior to *COSMOS*. Note that, in POAS, we conduct this preprocessing only once as our online aggregation is read-mostly application.

Table 2. Performance of Preprocessing

Systems	Processing Time (sec)				
	PS=20G	PS=40G	PS=60G	PS=80G	PS=100G
COSMOS	3840.39	6578.42	12539.12	17383.56	31254.26
POAS	1745.63	3289.29	5641.85	6953.47	8978.96

7 Conclusions

In this paper, we point out the limitations of online aggregation for skewed data distribution and present two fundamental principles that need to be considered in the problem solution. Based on the two principles, we propose POAS, a partition-based online aggregation system that (a) organizes dataset as a set of partitions, which is indexed by NRB-T, to prune unneeded data for each query, (b) shuffles each partition to gain well randomness of tuples so that we can use sequentially scan instead of random access to reduce the I/O cost during sampling, (c) collects unbiased sample from overlapped partitions with dynamic sample size to archive appropriate sample proportion. Finally, we have evaluated POAS on the TPC-H benchmark for skew data distribution, and the results demonstrate the efficiency and effectiveness of our approach.

Acknowledgments. This work is supported by National Key Basic Research program of China under Grants No. 2010CB328104, National Natural Science Foundation of China under Grants No. 61070161, No. 61070158, No. 61003257, No. 60903161 and No. 60903162, China National Key Technology R&D Program under Grants No. 2010BAI88B03, China Specialized Research Fund for the Doctoral Program of Higher Education under Grants No. 200802860031 and No. 20110092130002, Jiangsu Provincial Natural Science Foundation of China under Grants No. BK200 8030, Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No. BM2003201, and Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grants No. 93K-9.

We would like to thank Sai Wu and his team for providing us with the COSMOS source code.

References

1. Wu, S., Ooi, B.C., Tan, K.L.: Continuous sampling for online aggregation over multiple queries. In: SIGMOD 2010, pp. 651–662. ACM, New York (2010)
2. Chaudhuri, S., Das, G., Datar, M., Motwani, R., Narasayya, V.: Overcoming limitations of sampling for aggregation queries. In: ICDE 2001, pp. 534–542 (2001)
3. Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online aggregation. SIGMOD Rec. 26, 171–182 (1997)
4. Haas, P.J.: Large-sample and deterministic confidence intervals for online aggregation. In: SSDBM 1997, pp. 51–63. IEEE Computer Society, Washington, DC, USA (1997)
5. Haas, P.J., Hellerstein, J.M.: Ripple joins for online aggregation. SIGMOD Rec. (1999)
6. Luo, G., Ellmann, C.J., Haas, P.J., Naughton, J.F.: A scalable hash ripple join algorithm. In: SIGMOD 2002 (2002)
7. Wu, S., Jiang, S., Ooi, B.C., Tan, K.L.: Distributed online aggregations. In: Proc. VLDB Endow. (2009)
8. Condie, T., Conway, N., Alvaro, P.: Hellerstein: Online aggregation and continuous query support in mapreduce. In: SIGMOD 2010 (2010)
9. Böse, J.H., Andrzejak, A., Högqvist, M.: Beyond online aggregation: parallel and incremental data mining with online map-reduce. In: MDAC 2010 (2010)
10. Pansare, N., Borkar, V., Jermaine, C., Condie, T.: Online aggregation for large mapreduce jobs. In: VLDB 2011, ACM, Seattle (2011)
11. Borkar, V., Carey, M., Grover, R., Onose, N., Vernica, R.: Hyracks: A flexible and extensible foundation for data-intensive computing. In: ICDE 2011, pp. 1151–1162 (2011)
12. Jacobs, A.: The pathologies of big data. Commun. ACM 52, 36–44 (2009)
13. Bowen, T.F., Gopal, G., Herman, G., Hickey, T., Lee, K.C., Mansfield, W.H., Raitz, J., Weinrib, A.: The datacycle architecture. Commun. ACM (1992)
14. Candea, G., Polyzotis, N., Vingralek, R.: A scalable, predictable join operator for highly concurrent data warehouses. In: Proc. VLDB Endow., vol. 2, pp. 277–288 (2009)
15. Chaudhuri, S., Narasayya, V.: Program for tpc-d data generation with skew, ftp://ftp.research.microsoft.com/pub/user/viveknar/tpcdskew

A Relational-Based Approach for Aggregated Search in Graph Databases*

Thanh-Huy Le[1], Haytham Elghazel[2], and Mohand-Saíd Hacid[1]

[1] Université de Lyon, CNRS
Université Lyon 1, LIRIS UMR 5205, F-69622, France
thanhhuy27@gmail.com, mohand-said.hacid@univ-lyon1.fr
[2] Université de Lyon, Laboratoire GAMA, 69622 Villeurbanne
haytham.elghazel@univ-lyon1.fr

Abstract. In this paper, we investigate the problem of assembling fragments from different graphs to build an answer to a user query. The goal is to be able to provide an answer, by aggregation, when a single graph cannot satisfy all the query constraints. We provide the underlying basic algorithms and a relational framework to support aggregated search in graph databases. Our objective is to provide a flexible framework for the integration of data whose structure is graph-based (e.g., RDF). The idea is that the user has not to specify a join operation between fragments. The way the fragments can be combined is a discovery process and rests on a specific algorithm. We also led some experiments on synthetic datasets to demonstrate the effectiveness of this approach.

Keywords: graph databases, relational databases, query processing, aggregated search.

1 Introduction

Database research has been facing a new challenge raised by the emergence of massive, complex structural data, in the form of sequences, trees, and graphs [18]. Graphs have become increasingly important in modelling complex structures and schemaless data in many application domains such as bioinformatics [24], chemistry, web, social networks [3], business processes [16], telecommunication, etc. For instance, graphs may represent molecular structures of chemical compounds in chemistry, the organization of entities for images in computer vision, the ER diagrams in database design, the UML diagrams in software engineering, and so on. In addition, Web sites, XML and RDF documents can also be modeled as graphs [5,2].

The database community has had a long-standing interest in querying graph databases [18]. Given a graph database $\boldsymbol{\mathcal{D}} = \{g_1, g_2, \ldots, g_n\}$ and a query graph q, the task is to retrieve one or several graphs from the database that are similar

[*] Research partially supported by Agence Nationale de la Recherche-ANR (project AOC) and Rhone-Alpes region (project Web Intelligence).

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 33–47, 2012.
© Springer-Verlag Berlin Heidelberg 2012

to the query, which is generally referred to as *graph matching*. Standard graph matching approaches are generally categorized as either exact or approximate [13]. When a mapping exists between both graphs (query graph q and data graph g_i), this is called an isomorphism, and q is said to be isomorphic to g_i.

The problem of graph query processing has been tackled recently and a lot of methods have been proposed [7,22,25,26,4,17,23]. The underlying techniques have mainly targeted quick retrieval of the graphs that are supposed to be the answer to a given query. For such approaches, the challenge is to answer the following questions : (1) How to efficiently compare two graphs and (2) How to reduce as much as possible the search space and then the number of pair-wise comparisons of graphs? These are usually tough problems in many real-world applications where graphs are extremely large.

Although the graph query problem has been tackled in the last decade, no attention has been paid to the problem of assembling graphs in a sensed way to provide an answer to a given query graph q if (1) no single candidate graph turns out to be isomorphic to q, or (2) additional answers to q are needed. For example, we may have a query whose objective is to find the profile information related to a professor such as her/his associated universities, her/his research interests, her/his research histories (e.g. projects, publications), some of her/his personal information, and so on. However, from a huge volume of documents such as academic databases, multimedia databases, yellow pages, etc., the required information is not only contained in a single document, but spread over several documents. In other words, one document contains her/his personal data, one contains her/his publications, another contains her/his affiliations, and so on. This issue is shown in Figure 1 and Figure 2 where given a query graph q and a set of three data graphs $D = \{g_1, g_2, g_3\}$, a possible answer to q could be given by the aggregation of fragments from the two graphs g_1 and g_3.

In view of this context, this problem seems to have similar intention as the problem of *approximate graph matching* which tries to discover all the graphs that approximately contain the query graph when no match for the latter can be found in the graph database [23]. However, aggregated search problem differs from the problem of substructure similarity search in the sense that it provides different exact solutions (instead of relaxed ones) to the query graph by assembling graphs as answers to the query such that the aggregated graph contains the query. The challenge in this scenario is to answer the following questions : (1) how to determine the participating graphs to the aggregation and (2) how to build such an aggregation?

Relying on the effectiveness and scalability of relational database management systems, we propose an approach intended to support the *graph aggregation* in the framework of query evaluation, using the relational model. We focus on two tasks: (1) study how to translate a graph database into a relational database and (2) investigate the available powerful performance of RDBMS to deploy an efficient technique for processing and boosting graph aggregated search.

This approach focuses on the *directed labeled graphs* (we refer to them as graphs in the rest of the paper). Our design for data aggregation is targeted to supplement distributed graph query processing in such a way that query approximation (via data aggregation) will be supported. Our ultimate goal (in teh future) is to investigate query aggregation in distributed graph databases.

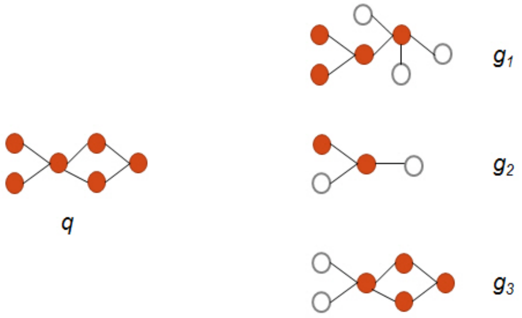


Fig. 1. A simple query graph q and three data graphs

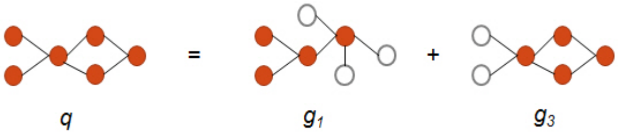


Fig. 2. An example of graph aggregation

The remainder of this paper is organized as follows: Section 2 gives the basic definitions used in this paper. Section 3 presents some available graph query processing algorithms and relational techniques that relate to our work. Section 4 introduces our approach for performing graph aggregated search on distributed graph databases using a relational infrastructure. Then, we demonstrate the efficiency and scalability of our technique by conducting some experiments on some synthetic graph databases in Section 5. We conclude in Section 6.

2 Preliminaries

This section introduces the terminology used in this paper and formally defines the problem. Conceptually, any kind of data can be represented by graphs. In labeled graphs, vertices and edges represent entities and relationships, respectively. The attributes associated with entities and relationships are called labels.

More formally, a *labeled graph* g is defined as a 6-tuple $(V, E, L_v, L_e, F_v, F_e)$ where V is the set of vertices; $E \subseteq V \times V$ is the set of edges joining two distinct vertices; L_v is the set of vertex labels; L_e is the set of edge labels; F_v is a function $V \rightarrow L_v$ that assigns labels to vertices and F_e is a function $E \rightarrow L_e$ that assigns labels to edges. The vertex set and the edge set of a graph g are denoted by $V(g)$ and $E(g)$, respectively. Labeled graphs are generally classified, according to the direction of their edges, into two main classes: *directed labeled graphs* such as XML and RDF and *undirected labeled graphs* such as social networks and chemical compounds. For example, the graphs shown in Figure 1 are undirected labeled graphs.

Definition 1 (Graph database). A graph database $\mathcal{D}$ is a collection of data graphs g_i where $\mathcal{D} = \{g_1, g_2, \ldots, g_n\}$.

Definition 2 (Candidate set). A candidate set $\mathcal{C}$ is a collection of data graphs from $\mathcal{D}$ that contain all the features appearing in a query graph q.

Definition 3 (Answer set). An answer set A is a collection of data graphs that are isomorphic to a query graph q.

Definition 4 (Non isomorphic set of graphs). A *non isomorphic set* $\mathcal{N}$ of graphs is a collection of data graphs from $\mathcal{C}$ that are not isomorphic to q.

Definition 5 (Graph aggregation problem). Given a query graph q and a non isomorphic set $\mathcal{N} = \{g_1, g_2, \ldots, g_m\}$, the problem of *graph aggregation query* is to find different subsets $\mathcal{S} = \{g_1, g_2, \ldots, g_k\}$ from $\mathcal{N}$ (*i.e.* $k \leq m$) for which the joining of fragments (subgraphs) $P_{g_1}, P_{g_2}, \ldots, P_{g_k}$ from graphs $g_1, g_2, \ldots, g_k$ respectively, leads to q, that is $q = (P_{g_1} \bowtie P_{g_2} \bowtie \ldots \bowtie P_{g_k})$. Here, the semantics of the join operation is the one used in the example given figures 1 and 2 to built a solution to the query of figure 1 by combining fragments stemming from the two graphs g_1 and g_3.

3 Graph Query Processing Algorithms

Graph matching [18,13] is considered to be one of the most complex issues because of its own combinatorial optimization problems. Typical classification in graph matching methods consists of exact graph matching and inexact graph matching. These methods allow to find the optimal solution but require exponential time and space due to the NP-completeness of the problem.

The problem of exact graph matching is particularly related to that of graph isomorphism, more clearly, if an edge resides between a pair of vertices (source vertex and destination vertex) in the query graph, then that edge must also reside between the corresponding pair in the data graph. In other words, one tries to retrieve an accurate pair-wise matching between vertices and edges of both graphs. The standard algorithm for graph and subgraph isomorphism detection is the one proposed by Ullman in [20].

The problem of graph query processing has been one of the hot topics discussed by the database community and a lot of methods have been proposed [7,22,25,26,4,17,15]. The underlying techniques have focused on quickly retrieving the graphs that are supposed to be the answer to a given query.

3.1 Approximate Graph Matching

Since exact matching is often too restrictive, approximate matching (similarity search) of complex structures becomes an important operation that must be supported efficiently. Instead of finding the data graphs that contain the query graph, these approximate matching methods find data graphs similar to the query graph. These kinds of queries are very useful within their own applications [11,21,12]. For example, a user may not know the exact composition of the full structure (s)he wants (*i.e.* query), but requires that it contains a set of small functional fragments. In such applications, it is possible to define an objective procedure that finds the similarity in the mapping between both graphs (query and data). A number of algorithms and models for approximate matching in graph databases were proposed (see, e.g., [12,9,10,23]).

3.2 Graph Aggregation for Query Processing Problem

The graph query processing has been widely considered for a long time. However, there is, so far, few of research on automatically assembling graphs such that a possible answer can be built from a combination of fragments of the graphs, in case no single isomorphic graph is found.

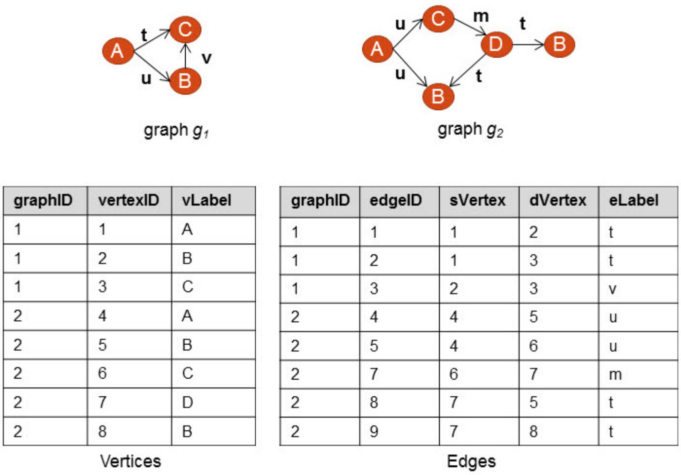
The problems raised by aggregation in the context of documents are discussed in [8]. However, the paper does not address formal and algorithmic issues. Motivated by this problem, Elghazel and Hacid [6] proposed an approach intended to support the graph aggregation in the framework of query processing. Their work, at first glance, seems to have similar intention as the problem of approximate matching as mentioned above. However, instead of trying to retrieve all the graphs that relatively contain the query as the approximate graph matching problem does, this approach returns several exact answers by combining graphs whose aggregation contains the query.

4 Relational-Based Approach for Aggregated Search in Distributed Databases

In this section, we discuss the relational-based mechanism used for graph aggregated search we propose.

4.1 Relational Encoding Schemes

Selection of an appropriate mapping schema for each graph member in the graph database is an important initial step in graph querying process. Graph structured databases can be stored by using various relational mapping schemas [14,1,19]. In this paper, we focus on the following two classes of encoding schemes initially used in [15]: (1) *Vertex-Edge* mapping scheme and (2) *Edge-Edge* mapping scheme. We also briefly show how to implement it in a standard Relational Database Management System. Although our approach refers to some fundamental techniques introduced in [15], it is different compared to [15] in the sense



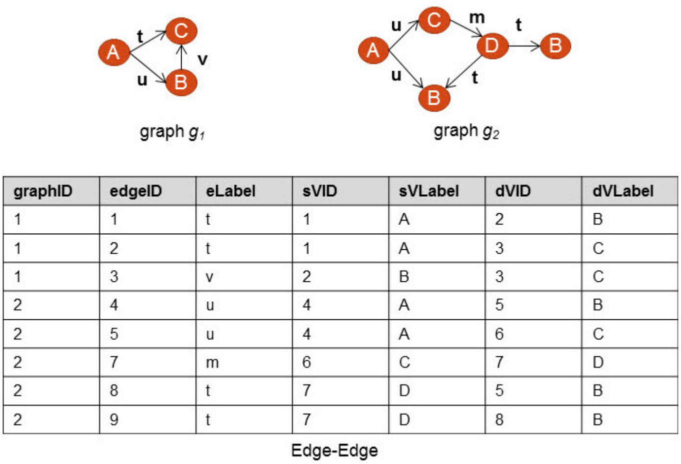
graphID	vertexID	vLabel
1	1	A
1	2	B
1	3	C
2	4	A
2	5	B
2	6	C
2	7	D
2	8	B

Vertices

graphID	edgeID	sVertex	dVertex	eLabel
1	1	1	2	t
1	2	1	3	t
1	3	2	3	v
2	4	4	5	u
2	5	4	6	u
2	7	6	7	m
2	8	7	5	t
2	9	7	8	t

Edges

Fig. 3. An example of Vertex-Edge mapping scheme for two graphs g_1 and g_2



graphID	edgeID	eLabel	sVID	sVLabel	dVID	dVLabel
1	1	t	1	A	2	B
1	2	t	1	A	3	C
1	3	v	2	B	3	C
2	4	u	4	A	5	B
2	5	u	4	A	6	C
2	7	m	6	C	7	D
2	8	t	7	D	5	B
2	9	t	7	D	8	B

Edge-Edge

Fig. 4. An example of Edge-Edge mapping scheme for two graphs g_1 and g_2

that while [15] performs the exact graph query mechanism, we will focus on the graph aggregated search issue in distributed graph databases.

In the *Vertex-Edge* scheme, each graph is assigned a unique *graphID*. Each vertex is identified by a unique numeric ID (*vertexID*) inside its container graph. Moreover, each vertex has its own *label* attribute. Each vertex is represented by one triple in a single table (*Vertices table*) in which all existing vertices are stored. Similarly, all edges in the graph database are stored in the second table

(*Edges table*) which not only contains the unique id of the edge (*edgeID*), the id of the container graph but also the id of the source vertex, the id of the destination vertex and the edge label. In summary, two tables of *Vertex-Edge* scheme have the following structures:

– *Vertices(<u>vertexID</u>, vertexLabel, graphID)*
– *Edges(<u>edgeID</u> , graphID, sVertex, dVertex, edgeLabel).*

Figure 3 shows an example of the *Vertex-Edge* encoding scheme of a graph database.

The second relational scheme, Edge-Edge encoding scheme has the following structure:

– *Edges(<u>edgeID</u>, graphID, eLabel, sVID, sVLabel, dVID, dVLabel).*

Each edge is described by a distinct tuple on which each tuple consists of the *graphID*, the IDs and the labels of the source and destination vertices. In addition, each edge is identified by a distinct numeric *edgeID*.

Figure 4 depicts an example of the *Edge-Edge* encoding scheme of a graph database.

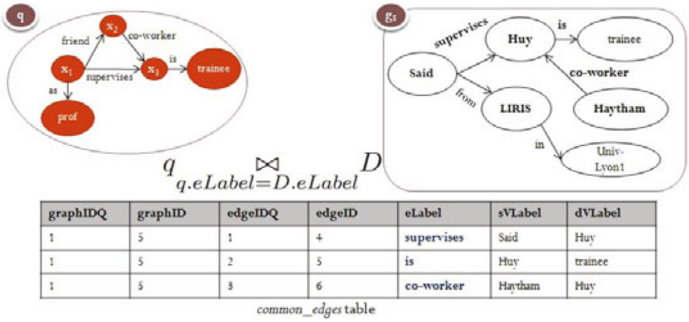
As stated in [15], the Edge-Edge scheme is regarded as a de-normalized form of the Vertex-Edge scheme. In other words, the Edge-Edge scheme is a uniform relation of all vertices and edges, thus it is considered to be more efficient for the querying issue (*i.e.* SQL SELECT statement). For instance, given a query graph q that holds m vertices and p edges, in order to retrieve that graph using *Vertex-Edge* scheme it requires $m+p$ join operations between *Vertices* table and *Edges* table. Meanwhile, by using the *Edge-Edge* scheme, one only needs p joins. However, the *Vertex-Edge* scheme shows its better performance in dealing with update operations (*i.e.* SQL INSERT, DELETE, UPDATE statements).

Our work mainly focuses on querying issue, thus we choose the *Edge-Edge* scheme as the encoding one for the query processing mechanism.

4.2 Common Edge Search

Edges or relationships are elements that carry the most important information in graphs. Hence, common edges search is the most basic function that plays a key role in our approach. This function discovers all edges that belong to both query graph q and a graph database $\mathcal{D}$. It also ensures that all edges in q are found in $\mathcal{D}$. As mentioned in previous sections, our technique is different from approximate graph matching problem in such a way that it leads to the calculation of different exact solutions to the query. Therefore, in case there does not exist a single edge in $\mathcal{D}$, we can immediately conclude that the answer is empty.

Common edges are the output of SQL-based join operation between a query and a graph database on their *eLabel* attribute. Figure 5 shows an example of common edges search between a query q and a graph database $\mathcal{D}$ consisting of one data graph g_5. Relying on the common edges table returned by this process we can conduct further search. The following SQL template describes the common edges search operation.



graphIDQ	graphID	edgeIDQ	edgeID	eLabel	sVLabel	dVLabel
1	5	1	4	supervises	Said	Huy
1	5	2	5	is	Huy	trainee
1	5	3	6	co-worker	Haytham	Huy

common_edges table

Fig. 5. An example of common edges between a query q and a graph database $\mathcal{D}$

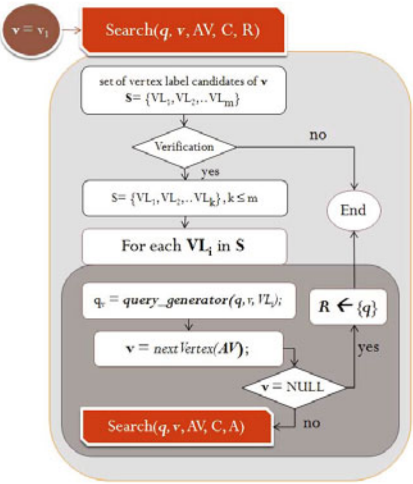


Fig. 6. Query evaluation process

SELECT q.edgeID, q.graphID,
graphID, eLabel, edgeID, sVLabel, dVLabel
FROM query q,
datagraphs g_1, datagraphs g_2, ..., datagraphs g_n
WHERE $\forall_{i=1}^{n}(g_i.eLabel = q.eLabel)$

where g_i is an instance of the table *datagraphs* and maps its edge labels to edge labels of the query q.

4.3 Query Graph Decomposition

In the following, we present the decomposition mechanism of a query graph for our aggregated search approach. First, given input query q, we split it into two

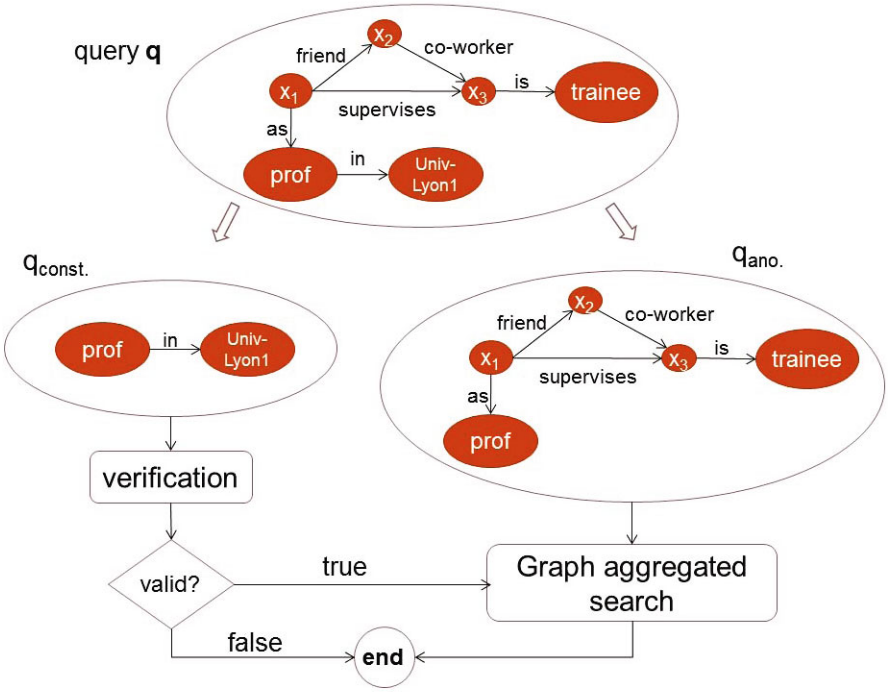


Fig. 7. Query decomposition

parts: constant part $q_{const.}$ (i.e. constant query or a set of edges linking two distinct labeled vertices) and anonymous part $q_{ano.}$ (i.e. anonymous query or set of edges linking unlabeled vertices to the others). Then, our searching process initially verifies the constant query. This verification phase checks whether each edge of $q_{const.}$ (including edge label, source vertex label and destination vertex) exists in the common edge database C. The search process terminates when no edge is found in C. Otherwise, it goes to the graph aggregated search phase for the anonymous query which is represented in subsection 4.4. Figure 7 summarizes our decomposition mechanism and processing flow on an example.

4.4 Aggregated Search for Anonymous Query Graphs

In this subsection we formulate an algorithm to identify the matching between a given anonymous query q and the aggregation of graphs in $\boldsymbol{\mathcal{D}}$. This problem is regarded as the problem of label identification for the blank nodes in q.

Let us assume that AV is a set of anonymous vertices (variables) of q, ordered by decreasing order of their degree.

In principle, the algorithm starts from the first unlabeled vertex v in AV (Figure 6). We retrieve all possible labels (*i.e.* the set S) of vertices in C that have the same edge labels with v (v is the source vertex or destination vertex, both cases are considered). Then, a verification step is used to eliminate invalid labels in S. For each remaining label in S, we generate a new query q_v by assigning this label to v, then we move to the next unlabeled vertex in AV and continue recursively the process of label determination and labeling for remaining vertices. When the query is totally labeled, it is considered as an answer and it is added to the final result.

The reason for which we construct AV and start from the first elements in AV is to improve the efficiency of the search process. The *degree* of a vertex v is the number of edges incident to it. So, the higher the degree is for a given vertex, the less possibility of label candidates returned for it, and from this point, we can initially eliminate as many as possible negative candidates (i.e. reduce the number of input candidates for the verification phase). This work also helps to decrease the number of recursions, and therefore, quickly reach the termination of the algorithm (in case of no valid candidate is found), or quickly reach the final result $\boldsymbol{\mathcal{A}}$.

The principle of our technique is summarized in the following steps:

1. In C, find a set of vertices S (label candidates) which have *similar edge labels* with v
2. If no vertex is found (i.e. $S = \emptyset$), then terminate the algorithm.
3. Remove from S invalid candidates (verification step).
4. If all candidates are removed (i.e $S = \emptyset$ after verification), then terminate the algorithm.
5. For each label V_L in S
   (a) generate a new query q_v from q by assigning V_L to v (i.e. v becomes a constant vertex)
   (b) move to next unlabeled vertex in AV (i.e. v = next vertex in AV)
   (c) If all vertices in S are labeled (i.e. $v = \emptyset$), a mapping is found. We add this mapping to the final result. Otherwise, a recursive call is performed by using q_v and the new vertex v as the input.

We clarify the verification phase in step 3. In q we retrieve a set of constant vertices $K_v = k_1, k_2, ..., k_m$ which are neighbors of v. Then, we search edges joining two vertices $e_i \subseteq v \times k_i$ in a set of common edges C. If there exists at least one e_i (i.e. $e_i.eLabel$, $v.vLabel$, $k_i.vLabel$) not found in C, in other words, if the count of the edge is equal to zero (count(edgeID) = 0), we can conclude that this candidate is invalid and it should be removed from S. In the case v is a source vertex, for each label candidate $VLabel$ of v, the verification is performed by the following SQL template:

SELECT count(edgeID)
FROM commonedges c
WHERE $c.sVLabel = v.VLabel$
AND $\forall_{i=1}^{m}(c.eLabel = e_i.eLabel)$
AND $\forall_{i=1}^{m}(c.dVLabel = k_i.VLabel)$

In the case v is a destination vertex, the SQL template is as follows:

Algorithm 1: AGASeach(q, v, AV, C)

```
Input: $q$ is a query graph; $AV$ ($AV \neq \emptyset$) is a set of unlabeled vertices
       (variables) in $q$ ordered by their degrees;
       $v$ is the in-process vertex in $AV$ ($v \in AV$). $C$ is a set of common
       edges between $q$ and set of data graphs $\mathcal{D}$.
Output: $\mathcal{A}$ is final answer set.

$S$ = findLabelCandidates($v[E_L], C$);
if ($S = \emptyset$) then
    break;
else
    $S$ = verification ($S$); // elimination of invalid labels
    if ($S = \emptyset$) then
        break;
    else
        foreach $V_L$ in $S$ do
            $q_v$ = query_generator($q, v, V_L$);
            $v$ = nextVertex($AV$);
            if ($v = \emptyset$) then
                $\mathcal{A} = \mathcal{A} \cup \{q\}$;
            else
                AGASeach($q_v, v, AV, C$); // recursive call
            end
        end
    end
end
```

SELECT count(edgeID)
FROM commonedges c
WHERE $c.dVLabel = v.VLabel$
AND $\forall_{i=1}^{m}(c.eLabel = e_i.eLabel)$
AND $\forall_{i=1}^{m}(c.sVLabel = k_i.VLabel)$

In step 5a, we create a new query from the current query by using the following SQL template:

CREATE TABLE new query q_v as SELECT * FROM query q;
UPDATE q_v SET $sVLabel = V_L$ WHERE $sVID = v$;
UPDATE q_v SET $dVLabel = V_L$ WHERE $dVID = v$;

In summary, our approach is algorithmically formed in Algorithm 1.

5 Performance Evaluation

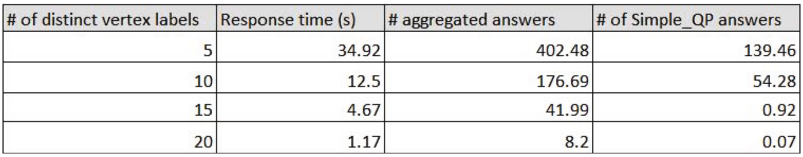
In this section, we report on first empirical results to evaluate the effectiveness and efficiency of our technique. Our experiments are conducted on synthetic datasets. We generate a large number of graphs by using our own graph generator module. This module allows us to specify the number of graphs (n), the number of distinct vertex labels (L_v), the number of distinct edge labels (L_e), the average number of edges for each graph ($|E|$) and the average number of vertices for each graph ($|V|$). For this experiment, we generate a set of graph datasets with different numbers of distinct vertex labels. The average number of vertices and edges in the data graphs are 20 and 30, respectively. The number of distinct edge labels is 20 while the number of distinct vertex labels varies from 5 to 20. There are 1000 data graphs and 100 queries in the considered experiments, that share the same number of distinct vertex labels and the number of distinct edge labels with data graphs. The average number of vertices and the average number of edges in each query are 10 and 15, respectively.

It is always a good choice to filter as many as possible negative graph candidates before going to the query processing step. However, in these experiments, we only focus on the performance test of the query processing as it is the core of our approach. In other words, the execution time of either this experiment and the others in this paper do not include the filtering phase which is independent from our approach in terms of input data graphs.

The idea in using relational database in storing and querying graph databases is to take advantage from their powerful features of scalability and efficiency. We examine the query performance of our SQL-based aggregated search approach. Since the parameters of synthetic datasets are adjustable, especially the number of distinct vertex labels, the overall performance of our approach is seriously not stable. In other words, the number of distinct vertex labels has a heavy impact on the capability and the scalability of our technique. For an interesting assess of the results gained with our aggregated search, we propose to measure the advantage of our approach to find new output solutions compared to the traditional subgraph isomorphism query approaches (denoted *Simple_QP* here). The average output size of *Simple_QP* is also recorded.

The experiments then examine 100 queries with a number of distinct vertex labels ranging from 20 down to 5. The execution results of both the approaches are summarized in Figure 8 and demonstrated by scatters in Figure 9. The X-axis represents the number of distinct vertex labels, the Y-axis shows the average answer set sizes, and the different curves represent both approaches.

As expected, the results of these experiments show that the response time and the average output size of our approach are strictly related with the number of distinct vertex label. Strictly speaking, there are two remarkable points: (1) the less number of distinct vertex labels, the more troublesome the aggregated search is, since the number of graph candidates increases, and then, the verification and query generator become costly, and (2) the more the number of distinct vertex labels, the less number of answers returned from the *Simple_QP* approaches, and therefore, the more interesting the aggregated search is. In fact, the experiments confirm the ability of our technique to deliver new answers that can improve the efficiency and the precision of a query processing system. For instance, when the number of distinct vertex labels is equal to 10, *Simple_QP* approach returns 54.28 solutions, meanwhile we obtain 176.69 solutions (*i.e.* 3 times more) with our approach. There is another remark that *Simple_QP* fails to return any answer for a lot of query graphs when the number of distinct vertex labels reaches 15, however, the aggregated search shows its diversity on query answers with almost 42 solutions.



# of distinct vertex labels	Response time (s)	# aggregated answers	# of Simple_QP answers
5	34.92	402.48	139.46
10	12.5	176.69	54.28
15	4.67	41.99	0.92
20	1.17	8.2	0.07

Fig. 8. Performance evaluation on varying the number of vertex labels

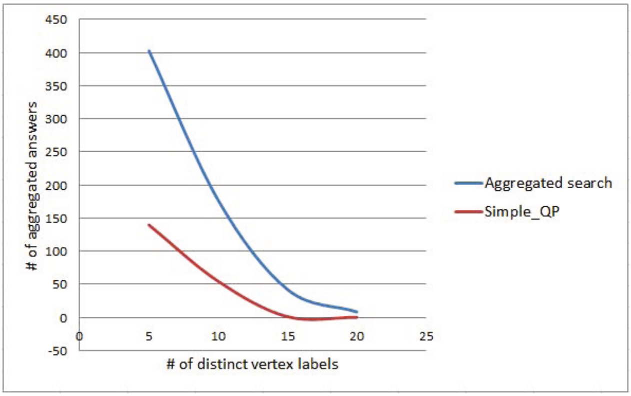


Fig. 9. Impact of the value of the number of distinct vertex label on both approaches

6 Conclusion

In this paper, we have discussed the problem of assembling graphs to provide answers to a given query graph in case no single candidate graph is isomorphic with the query. We also presented a relational technique for supporting the graph aggregated search. The approach is developed on top of relational infrastructure by using pure SQL scripts in the form of stored procedures (views) and functions, thus absolutely residing in a RDBMS. We proposed a relational encoding scheme to encode graph databases and translate graph queries into SQL queries. Then, we retrieved common edges between query graph and data graphs. From this set of data, we performed graph aggregated search on queries with anonymous nodes. Finally, through a set of experiments on a synthetic dataset we obtained preliminary results about the efficiency and scalability of our approach. The evaluation results indicate that our relational technique for processing graph aggregated search is strictly correlated with the number of distinct vertex labels. This point needs additional investigation. Moreover, the experimental results also show that the execution time for the aggregated search is rather high. Therefore, we believe that there is a need for additional work for improving and optimizing the performance of our approach for storing and querying large graph databases.

Regarding our future work, we will deploy more experiments on different datasets with other types of graphs. We will also conduct database pre-processing module in which we construct a summary layer of the underlying graph database. Then, relying on this layer we apply filtering phase to eliminate as many as possible negative graph members which are certainly not in the answer set. The pre-processing phase is very useful in reducing search space and improving performance or the efficiency of our SQL-based mechanism for the aggregated graphs search approach.

References

1. Balmin, A., Papakonstantinou, Y.: Storing and querying xml data using denormalized relational databases. The VLDB Journal 14(1), 30–49 (2005)
2. Bonstrom, V., Hinze, A., Schweppe, H.: Storing rdf as a graph
3. Cai, D., Shao, Z., He, X., Yan, X., Han, J.: Community Mining from Multi-Relational Networks. In: Jorge, A.M., Torgo, L., Brazdil, P.B., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, pp. 445–452. Springer, Heidelberg (2005)
4. Cheng, J., Ke, Y., Ng, W., Lu, A.: Fg-index: towards verification-free query processing on graph databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 857–872 (2007)
5. Dau, F.: RDF as Graph-Based, Diagrammatic Logic. In: Esposito, F., Raś, Z.W., Malerba, D., Semeraro, G. (eds.) ISMIS 2006. LNCS (LNAI), vol. 4203, pp. 332–337. Springer, Heidelberg (2006)
6. Elghazel, H., Hacid, M.-S.: Aggregated Search in Graph Databases: Preliminary Results. In: Jiang, X., Ferrer, M., Torsello, A. (eds.) GbRPR 2011. LNCS (LNAI), vol. 6658, pp. 92–101. Springer, Heidelberg (2011)
7. Giugno, R., Shasha, D.: Graphgrep: A fast and universal method for querying graphs. In: Proceedings of the International Conference on Pattern Recognition, pp. 112–115 (2002)
8. Kopliku, A., Pinel-Sauvagnat, K., Boughanem, M.: Aggregated search: potential, issues and evaluation. Technical Report RT2009-4FR, IRIT (2009), http://www.irit.fr/PERSONNEL/SIG/kopliku/
9. Neuhaus, M., Bunke, H.: Self-organizing maps for learning the edit costs in graph matching. IEEE Transactions on Systems, Man, and Cybernetics, Part B 35(3), 503–514 (2005)
10. Neuhaus, M., Bunke, H.: Automatic learning of cost functions for graph edit distance. Information Sciences 177(1), 239–247 (2007)

11. Petrakis, E.G.M., Faloutsos, C.: Similarity searching in medical image databases. IEEE Transactions on Knowledge and Data Engineering 9(3), 435–447 (1997)
12. Raymond, J.W., Gardiner, E.J., Willett, P.: Calculation of graph similarity using maximum common edge subgraphs. The Computer Journal 45, 631–644 (2002)
13. Riesen, K., Jiang, X., Bunke, H.: Exact and inexact graph matching: Methodology and applications. In: Aggarwal, C.C., Wang, H. (eds.) Managing and Mining Graph Data, pp. 217–247. Springer, US (2010)
14. Sakr, S.: Storing and Querying Graph Data using Efficient Relational Processing Techniques. In: Yang, J., Ginige, A., Mayr, H.C., Kutsche, R.-D. (eds.) Information Systems: Modeling, Development, and Integration. LNBIP, vol. 20, pp. 379–392. Springer, Heidelberg (2009)
15. Sakr, S., Al-Naymat, G.: Efficient relational techniques for processing graph queries. Journal of Computer Science and Technology 25(6), 1237–1255 (2010)
16. Sakr, S., Awad, A.: A framework for querying graph-based business process models. In: Proceedings of the 19th International Conference on World Wide Web (WWW), pp. 1297–1300. ACM, New York (2010)
17. Shang, H., Zhang, Y., Lin, X., Yu, J.X.: Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. In: Proceedings of the International Conference on Very Large Data Bases, pp. 364–375 (2008)
18. Shasha, D., Wang, J.T.L., Giugno, R.: Algorithmics and applications of tree and graph searching. In: Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), pp. 39–52 (2002)
19. Tatarinov, I., Viglas, S.D., Beyer, K., Shanmugasundaram, J., Shekita, E., Zhang, C.: Storing and querying ordered xml using a relational database system. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, pp. 204–215 (2002)
20. Ullmann, J.R.: An algorithm for subgraph isomorphism. Journal of ACM 23(1), 31–42 (1976)
21. Willett, P., Barnard, J.M., Downs, G.M.: Chemical similarity searching. Journal of Chemical Information and Computer Sciences 38(6), 983–996 (1998)
22. Yan, X., Yu, P.S., Han, J.: Graph indexing: A frequent structure-based approach. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 335–346 (2004)
23. Yan, X., Yu, P.S., Han, J.: Substructure similarity search in graph databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 766–777 (2005)
24. Yang, Q., Sze, S.-H.: Path matching and graph matching in biological networks. Journal of Computational Biology 14(1), 56–67 (2007)
25. Zhang, S., Hu, M., Yang, J.: Treepi: A novel graph indexing method. In: Proceedings of the International Conference on Data Engineering, pp. 966–975 (2007)
26. Zhao, P., Yu, J.X., Yu, P.S.: Graph indexing: Tree + delta >= graph. In: Proceedings of the International Conference on Very Large Data Bases, pp. 938–949 (2007)

Discovery of Keys from SQL Tables

Van Bao Tran Le[1], Sebastian Link[2], and Mozhgan Memari[1]

[1] School of Information Management, Victoria University of Wellington, New Zealand
[2] Department of Computer Science, University of Auckland, New Zealand

Abstract. Keys play a fundamental role in all data models. They allow database systems to uniquely identify data items, and therefore promote efficient data processing in most applications. Due to this role support is required to discover keys. These include keys that are semantically meaningful for the application domain, or are satisfied by a given database instance. Here, we study the discovery of keys from SQL tables. We investigate structural and computational properties of Armstrong tables for sets of SQL keys that are currently perceived as semantically meaningful. Inspections of Armstrong tables enable data engineers to consolidate their understanding of the semantics of the application domain, and communicate this understanding to other stake-holders of the database, e.g. domain experts or managers. The stake-holders may want to make changes to the tables or provide entirely different tables in order to communicate their expert views to the data engineers. For such purpose we propose data mining algorithms that discover keys from a given SQL table. Finally, we define formal measures to assess the distance between sets of SQL keys. The measures can be applied to empirically validate the usefulness of Armstrong tables, and to automate marking and feedback of non-multiple choice questions in database courses.

1 Introduction

Context. Keys play a fundamental role in understanding both the structure and semantics in databases. Given an SQL table schema, a key is a collection of columns whose values uniquely identify rows. That is, no two rows have matching total values in each of the key columns. The concept of a key is essential for many other data models, including semantic models, object models and XML. The discovery of semantically meaningful SQL keys is a crucially important task in many areas of modern data management, e.g., data modeling, database design, query optimization, indexing, and data integration. This paper is concerned with methods for semi-automated schema-driven as well as automated data-driven SQL key discovery. There has been a great demand on the part of industry for such methods, because they vastly simplify the job of the DBA and thereby decrease the overall cost of database ownership. The discovery of composite keys is especially difficult, because the number of possible keys increases exponentially with the number of columns. Because such functionality is nevertheless needed by industry, the goal is to provide practical algorithms that have good typical case behavior. Industry-leading data modeling tools, such as *ERwin*, emphasize

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 48–62, 2012.
© Springer-Verlag Berlin Heidelberg 2012

the need for good test data to validate the semantics of the models they produce [4]. Intuitively, this calls for the generation of test data which satisfy the keys perceived semantically meaningful and violate the keys perceived meaningless.

Running Example. Consider a simple database that collects basic information about the weekly schedule of courses. That is, we have a schema SCHEDULE with columns *C_ID*, *L_Name*, *Time* and *Room*. The schema stores the time (including the weekday) and room in which a lecturer (identified by their L_Name) gives a course (identified by their C_ID). An SQL definition may look as follows:

```
CREATE TABLE SCHEDULE (
        C_ID CHAR[5], L_Name VARCHAR,
        Time CHAR[15], Room VARCHAR,
        PRIMARY KEY (C_ID, Time);)
```

The table schema specifies additional assertions. The primary key forces rows over SCHEDULE to be `NOT NULL` in the *C_ID* and *Time* columns, and to be unique on their $\{C\_ID, Time\}$ projections (no two distinct rows must have the same value in both the *C_ID* column and the *Time* column). A team of data engineers may wonder if the semantics of the application domain has been captured. They decide to generate some good test data to discuss their current understanding with the domain experts. In fact, a joint inspection of the data sample in Table 1 reveals some concern about rows 1 and 3. As a consequence, the team decides to specify the uniqueness constraint (UC) $u(Time,L\_Name,Room)$. They produce the data sample on the left of Table 2 to consolidate their new understanding. Rows 1 and 3, and 1 and 4, respectively, reveal that the UC $u(Time,L\_Name,Room)$ should be replaced by the two stronger UCs $u(Time,L\_Name)$ and $u(Time,Room)$. The example shows the potential benefit of investigating good sample data for the discovery of semantically meaningful SQL keys. As a revised SQL table schema the team specifies

Table 1. An Armstrong table for the SQL table schema SCHEDULE

C_ID	Time	L_Name	Room
11301	Mon, 10am	Ullman	Red
11301	Tue, 02pm	Ullman	Red
78200	Mon, 10am	Ullman	Red
99120	Wed, 04pm	`ni`	`ni`

Table 2. Armstrong tables for revised constraint sets

C_ID	Time	L_Name	Room
11301	Mon, 10am	Ullman	Red
11301	Tue, 02pm	Ullman	Red
78200	Mon, 10am	Ullman	`ni`
99120	Mon, 10am	`ni`	Red

C_ID	Time	L_Name	Room
11301	Mon, 10am	Ullman	Red
11301	Tue, 02pm	Ullman	Red
78200	Mon, 10am	**Fagin**	**Blue**
99120	Wed, 04pm	`ni`	`ni`

```
CREATE TABLE SCHEDULE' (
        C_ID CHAR[5], L_Name VARCHAR,
        Time CHAR[15], Room VARCHAR,
        PRIMARY KEY (C_ID, Time),
        UNIQUE (L_Name, Time), UNIQUE (Room, Time);)
```

While this approach appears to be beneficial it requires us to address our first research question: what constitutes good test data, and how to create it automatically? Domain experts are likely to welcome an opportunity to modify values in the test data to reflect their domain knowledge, or provide entire test data themselves. In this or similar situations, the data engineers need automated means to discover a representation of the keys that are satisfied by the test data. For example, when the domain experts inspect Table 1, they may simply suggest to use the right data sample in Table 2 instead, with the updated values indicated by bold font. On input of this table, a constraint mining algorithm would return `NOT NULL` constraints on *C_ID* and *Time*, and the three UCs u(*C_ID, Time*), u(*L_Name, Time*), and u(*Room, Time*). This, again, leads to the definition of SCHEDULE′. This motivates our second research question: how can we discover the SQL keys that are satisfied by a given SQL table?

Contributions. In this paper we will establish detailed answers to the two questions above. As our first main contribution we investigate the well-known concept of Armstrong databases for the class of SQL keys. Armstrong databases formalize the concept of good test data in the sense that they satisfy the set Σ of keys currently perceived semantically meaningful, and violate all keys that are not implied by Σ. We characterize when a given SQL table is Armstrong with respect to a given set Σ of SQL keys. This characterization allows us to establish an algorithm that generates good test data for arbitrary sets of SQL keys. While we show that the problem of computing such Armstrong tables is precisely exponential in the number of column headers, our algorithm produces an Armstrong table whose size is at most quadratic as that of a minimum-sized Armstrong table. As a second main contribution we establish two algorithms that compute the set of minimal SQL keys satisfied by a given SQL table. While the problem requires generally exponential time, our algorithms show good best case behavior. As the final contribution we define formal measures that can be applied to i) empirically validate the usefulness of our Armstrong tables for the acquisition of semantically meaningful SQL keys, and ii) automate feedback and marking of database exam questions.

Organization. We summarize related work in Section 2, and give preliminary definitions in Section 3. We investigate structural and computational properties of Armstrong tables for the class of SQL keys in Section 4. In Section 5 we study the SQL key mining problem. Our formal measures of usefulness and their applications are discussed Section 6. We conclude in Section 7 where we briefly comment on future work, too.

2 Related Work

Data dependencies have been studied thoroughly in the relational model of data, cf. [1,9]. Dependencies are essential to the design of the target database, the maintenance of the database during its lifetime, and all major data processing tasks [1]. These applications also provide a strong motivation for developing data mining algorithms to discover the data dependencies that are satisfied by a given database. Armstrong databases are a useful design aid for data engineers that can help with the consolidation of data dependencies [13], the design of databases [14] and the creation of concise test data [6].

In the relational model, keys are subsumed by functional dependencies (FDs). Structural and computational problems of Armstrong relations have been investigated in the relational model for the class of keys [7], and the more general class of FDs [3,14]. The mining of keys and FDs in relations has also received considerable attention in the relational model [11,15,16]. However, this work has not considered occurrences of duplicate rows, null markers and `NOT NULL` constraints which are present in most SQL databases.

One very significant extension of Codd's basic relational model [5] is incomplete information [12,17]. This is mainly due to the high demand for the correct handling of such information in real-world applications. Approaches to deal with incomplete information comprise incomplete relations, or-relations or fuzzy relations. In this paper we focus on incomplete bags, and the most popular interpretation of a null marker as "no information" [2,17]. This is the general case of SQL tables where duplicate rows and null markers are permitted to occur in columns that are specified as null-able. Relations are idealized special SQL tables where no duplicate rows can occur and all columns are specified `NOT NULL`. Recently, Armstrong tables have been investigated for the combined class of keys and FDs [10]. In the present paper, we first establish non-trivial optimizations for Armstrong tables that arise from the focus on the sole class of keys. This provides insight into the trade-off between the expressiveness of data dependency classes and the efficiency of generating Armstrong tables. Intuitively, the focus on the less expressive class of keys results in smaller Armstrong tables which can be computed more efficiently. To the best of the authors' knowledge, no previous work has addressed the mining of keys in SQL tables, which could be due to the previous lack of a theoretical model. We are also unaware of any measures that capture the difference between sets of keys, nor their utilization for assessing the usefulness of Armstrong tables or database exam questions.

3 The SQL Table Model

Let $\mathfrak{H} = \{H_1, H_2, \ldots\}$ be a countably infinite set of symbols, called *column headers* or *headers* for short. A *table schema* is a finite non-empty subset T of $\mathfrak{H}$. Each header H of a table schema T is associated with an infinite domain $dom(H)$ of the possible values that can occur in column H. To encompass partial information every column can have a null marker, denoted by $\texttt{ni} \in dom(H)$. The intention of `ni` is to mean "no information".

For header sets X and Y we may write XY for $X \cup Y$. If $X = \{H_1, \ldots, H_m\}$, then we may write $H_1 \cdots H_m$ for X. In particular, we may write simply H to represent the singleton $\{H\}$. A *row* over T (T-row or simply row, if T is understood) is a function $r : T \to \bigcup_{H \in T} dom(H)$ with $r(H) \in dom(H)$ for all $H \in T$. The null marker occurrence $r(H) = \texttt{ni}$ associated with a header H in a row r means that there is no information about $r(H)$. That is, $r(H)$ may not exist or $r(H)$ exists but is unknown. For $X \subseteq T$ let $r(X)$ denote the restriction of the row r over T to X. A *table* t over T is a finite multi-set of rows over T. For a row r over T and a set $X \subseteq T$, r is said to be X-total if for all $H \in X$, $r(H) \neq \texttt{ni}$. Similar, a table t over T is said to be X-total, if every row r of t is X-total. A table t over T is said to be a *total table* if it is T-total.

A *uniqueness constraint* (UC) over a table schema T is an $u(X)$ where $X \subseteq T$. A table t over T is said to *satisfy* the UC $u(X)$ over T ($\models_t u(X)$) if for all $r_1, r_2 \in t$, if $r_1(X) = r_2(X)$ and r_1, r_2 are X-total, then $r_1 = r_2$. The semantics is that of SQL's UCs, and it reduces to that of a key for total tables [1].

Following Atzeni and Morfuni [2] a *null-free subschema* (NFS) over the table schema T is a an expression $\mathit{nfs}(T_s)$ where $T_s \subseteq T$. The NFS T_s over T is satisfied by a table t over T, denoted by $\models_t \mathit{nfs}(T_s)$, if and only if t is T_s-total. SQL allows the specification of column headers as `NOT NULL`. NFSs occur in everyday database practice: the set of headers declared `NOT NULL` forms the single NFS over the underlying table schema.

In schema design and maintenance data dependencies are normally specified as semantic constraints on the tables intended to be instances of the schema. During the design process or the lifetime of a database one usually needs to determine further dependencies which are implied by the given ones. Let T be a table schema, $\mathit{nfs}(T_s)$ an NFS, and $\Sigma \cup \{\varphi\}$ be a set of UCs over T. We say that Σ *implies* φ in the presence of $\mathit{nfs}(T_s)$, denoted by $\Sigma \models_{T_s} \varphi$, if every T_s-total table t over T that satisfies Σ also satisfies φ. If Σ does not imply φ in the presence of $\mathit{nfs}(T_s)$ we may also write $\Sigma \not\models_{T_s} \varphi$. Let $\Sigma^*_{T_s} = \{\varphi \mid \Sigma \models_{T_s} \varphi\}$ be the *semantic closure* of Σ. If we do not want to emphasize the presence of the NFS $\mathit{nfs}(T_s)$ we may simply write $\Sigma \models \varphi$, $\Sigma \not\models \varphi$ or Σ^*, respectively. The next lemma explains why minimal UCs are important. Indeed, for a set $\Sigma \cup \{u(X)\}$ of UCs, and an NFS $\mathit{nfs}(T_s)$ over T we call $u(X)$ *minimal* if and only if $\Sigma \models_{T_s} u(X)$ and for all $u(Y)$ over T where $\Sigma \models_{T_s} u(Y)$ and $Y \subseteq X$ we have $Y = X$.

Lemma 1. *Let T be a table schema, $\mathit{nfs}(T_s)$ an NFS, and $\Sigma \cup \{\varphi\}$ be a set of UCs over T. Then Σ implies φ in the presence of $\mathit{nfs}(T_s)$ if and only if there is some $u(Y) \in \Sigma$ such that $Y \subseteq X$.*

Proof. Let t be the table over T consisting of two rows r_1 and r_2 over T where r_1 and r_2 are XT_s-total and $r_1(X) = r_2(X)$, and $r_1(H) = \texttt{ni} = r_2(H)$ for all $H \in T - XT_s$, and $r_1(H) \neq r_2(H)$ for all $H \in (T - X) \cap T_s$. It follows that t is T_s-total and satisfies Σ, but t violates $u(X)$. □

Example 1. We can capture the SQL table schema of the running example as the table schema SCHEDULE $= \{CTLR\}$ with SCHEDULE$_s$ $= \{CT\}$. Let Σ consist of the two UCs $u(CT)$ and $u(LTR)$. It follows that Σ does not imply any of the following UCs: $u(CLR)$, $u(LT)$ nor $u(TR)$. For instance, the SCHEDULE$_s$-total table on the left of Table 2 satisfies Σ and violates every of the three UCs. As an application of Lemma 1 we see that neither of CLR, LT nor TR is a superset of CT or LTR. □

4 Discovering Keys from Armstrong Tables

In this section we investigate structural and computational properties of suitable data to test the semantic meaningfulness of uniqueness constraints over SQL table schemata. For this purpose, we use Armstrong tables to formalize the notion of suitable test data. Having introduced the concepts of strong agree sets and anti-keys, we characterize when an arbitrarily given SQL table is Armstrong for an arbitrarily given set of uniqueness constraints. The characterization is then used to compute Armstrong tables. Finally, we derive results on the time and space complexity associated with the computation of Armstrong tables.

4.1 Key Concepts

The formal concept of an *Armstrong database* was originally introduced by Fagin [9]. We require our tables to be Armstrong with respect to uniqueness constraints and the NFS. Intuitively, an Armstrong table satisfies the given constraints and violates the constraints in the given class that are not implied by the given constraints. This results in the following definition.

Definition 1. *Let Σ be a set of UCs, and nfs(T_s) an NFS over table schema T. A table t over T is said to be* Armstrong *for Σ and nfs(T_s) if and only if i) t satisfies Σ, ii) t violates every $\varphi \notin \Sigma^*_{T_s}$, iii) t is T_s-total, and iv) for every $H \in T - T_s$, t is not H-total.* □

Example 2. Let SCHEDULE $= \{CTLR\}$ with SCHEDULE$_s = \{CT\}$. Let Σ consist of the two UCs $u(CT)$ and $u(LTR)$. The left sample in Table 2 is Armstrong for Σ and SCHEDULE$_s$. For example, it violates the UCs: $u(CLR)$, $u(LT)$ and $u(TR)$, as well as every NFS *nfs*(T'_s) where T'_s is not contained in SCHEDULE$_s$. □

A natural question to ask is how we can characterize the structure of tables that are Armstrong. For this purpose we introduce the formal notion of strong agree sets for pairs of distinct rows, and tables.

Definition 2. *For two rows r_1 and r_2 over table schema T where $r_1 \neq r_2$ we define the* strong agree set *of r_1 and r_2 as the set of all column headers over T on which r_1 and r_2 have the same total value, i.e., $ag^s(r_1, r_2) = \{H \in T \mid r_1(H) = r_2(H)$ and $r_1(H) \neq \boldsymbol{ni} \neq r_2(H)\}$. Furthermore, the strong agree set of a table t over table schema T is defined as $ag^s(t) = \{ag^s(r_1, r_2) \mid r_1, r_2 \in t \wedge r_1 \neq r_2\}$.* □

Example 3. Let SCHEDULE $= \{CTLR\}$ with SCHEDULE$_s = \{CT\}$. Let Σ consist of the two UCs $u(CT)$ and $u(LTR)$. Let t denote the left sample in Table 2. The strong agree set of t consists of CLR, LT, TR, L, R, and T. □

For a table t to be Armstrong for Σ and $nfs(T_s)$, t must violate all uniqueness constraints $u(X)$ not implied by Σ in the presence of $nfs(T_s)$. Instead of violating all uniqueness constraints it suffices to violate those ones that are maximal with the property that they are not implied by Σ in the presence of $nfs(T_s)$. This motivates the following definition.

Definition 3. *Let Σ be a set of UCs, and $nfs(T_s)$ an NFS over table schema T. The set Σ^{-1} of all anti-keys with respect to Σ and $nfs(T_s)$ is defined as $\Sigma^{-1} = \{a(X) \mid X \subseteq T \wedge \Sigma \not\models_{T_s} u(X) \wedge \forall H \in (T - X)(\Sigma \models_{T_s} u(XH))\}$.* □

Hence, an anti-key for Σ is given by a maximal set of column headers which does not form a uniqueness constraint implied by Σ.

Example 4. Let SCHEDULE $= \{CTLR\}$ with SCHEDULE$_s = \{CT\}$. Let Σ consist of the two UCs $u(CT)$ and $u(LTR)$. The set Σ^{-1} of anti-keys with respect to Σ and SCHEDULE$_s$ consists of $a(CLR)$, $a(LT)$ and $a(TR)$. □

4.2 Structure of Armstrong Tables

The concepts from the last sub-section are sufficient to establish a characterization of Armstrong tables for the class of UCs over SQL table schemata.

Theorem 1. *Let Σ a set of UCs, and $nfs(T_s)$ an NFS over the table schema T. For all tables t over T it holds that t is an Armstrong table for Σ and $nfs(T_s)$ if and only if the following three conditions are satisfied:*

1. *for all $a(X) \in \Sigma^{-1}$ we have $X \in ag^s(t)$,*
2. *for all $u(X) \in \Sigma$ and for all $Y \in ag^s(t)$ we have $X \not\subseteq Y$,*
3. *$total(t) = \{H \in T \mid \forall r \in t(r(H) \neq \mathtt{ni})\} = T_s$.*

Proof. We show first that the three conditions are sufficient for t to be an Armstrong table for Σ and $nfs(T_s)$. Suppose that t is such that the three conditions are satisfied. It follows immediately from the last condition that t satisfies $nfs(T_s')$ if and only if $T_s' \subseteq T_s$. Let $u(X) \in \Sigma$. If there were two rows $r_1, r_2 \in t$ such that $r_1 \neq r_2$ and $X \subseteq ag^s(r_1, r_2) = Y$, then $Y \in ag^s(t)$. This, however, would violate the second condition. Hence, t satisfies $u(X)$. Since $u(X) \in \Sigma$ was an arbitrary choice we conclude that t satisfies Σ. Let $u(Y) \notin \Sigma^*$. Then there is some $a(X) \in \Sigma^{-1}$ such that $Y \subseteq X$ holds. From the first condition we conclude that $Y \subseteq ag^s(r_1, r_2)$ for some $r_1, r_2 \in t$ with $r_1 \neq r_2$. Consequently, t violates every uniqueness constraint not implied by Σ.

It remains to show that the three conditions hold necessarily whenever t is an Armstrong table for Σ and $nfs(T_s)$. Suppose that t is an Armstrong table for Σ and $nfs(T_s)$. The last condition follows immediately from the fact that t satisfies $nfs(T_s')$ if and only if $T_s' \subseteq T_s$. Since t satisfies Σ there cannot be any $Y \in ag^s(t)$ and $u(X) \in \Sigma$ such that $X \subseteq Y$ holds. We conclude that the second condition is satisfied. It remains to show that the first condition is satisfied, too. Let $a(X) \in \Sigma^{-1}$. We need to show that $X \in ag^s(t)$. From $a(X) \in \Sigma^{-1}$ it follows that $u(X) \notin \Sigma^*$. As t violates $u(X)$ it follows that there are $r_1, r_2 \in t$ such that $r_1 \neq r_2$ and $X \subseteq Y = ag^s(r_1, r_2)$. Also, from $a(X) \in \Sigma^{-1}$ it follows that for all $H \in T - X$ we have $u(XH) \in \Sigma^*$, and thus that t satisfies $u(XH)$. Suppose that there is some $H \in Y - X$. Then t satisfies $u(XH)$ and, therefore, $r_1(H) = \texttt{ni} = r_2(H)$ or $r_1(H) \neq r_2(H)$. This, however, is a contradiction as $H \in Y = ag^s(r_1, r_2)$. Consequently, $X = Y \in ag^s(t)$. □

Example 5. Let $\textsc{Schedule} = \{CTLR\}$ with $\textsc{Schedule}_s = \{CT\}$. Let Σ consist of the two UCs $u(CT)$ and $u(LTR)$, and let t denote the sample on the left of Table 2. Recall from the previous examples that $\Sigma^{-1} = \{a(CLR), a(LT), a(TR)\}$, and $ag^s(t) = \{CLR, LT, TR, L, R, T\}$. Since t satisfies the three conditions of Theorem 1, it follows that t is an Armstrong table for Σ and $\textsc{Schedule}_s$. □

4.3 Computation of Armstrong Tables

We will now use the characterization of Theorem 1 to compute Armstrong tables for an arbitrarily given set Σ of UCs and an arbitrarily given NFS $nfs(T_s)$ over an arbitrarily given table schema T. A great part of the computation is devoted to determine the set Σ^{-1}. For this purpose, we borrow concepts from hyper-graphs. Indeed, to compute Σ^{-1} we generate the simple hyper-graph $\mathcal{H} = (V, E)$ with vertex set $V = T$ and the set $E = \{X \mid u(X) \in \Sigma\}$ of hyper-edges. In fact, based on Lemma 1 we assume without loss of generality that Σ consists of minimal UCs only. If not, then we remove all those UCs from Σ that are not minimal. From this we obtain $\Sigma^{-1} = \{a(T - X) \mid X \in Tr(\mathcal{H})\}$ where $Tr(\mathcal{H})$ denotes the minimal transversals of the hyper-graph $\mathcal{H}$, i.e. the set of minimal sets X of T that have a non-empty intersection with each hyper-edge of $\mathcal{H}$ [8].

Lemma 2. *Let Σ be a set of UCs over table schema T. Then $\Sigma^{-1} = \{a(T-X) \mid X \in Tr(\mathcal{H})\}$.*

Proof. Recall that $Tr(\mathcal{H}) = \{X \subseteq T \mid \forall u(Z) \in \Sigma(Z \cap X \neq \emptyset) \wedge (\exists Y \subseteq X(\forall u(Z) \in \Sigma(Z \cap Y \neq \emptyset)) \Rightarrow Y = X)\}$.

We show first that if $X \in Tr(\mathcal{H})$, then $a(T - X) \in \Sigma^{-1}$. First it follows that $\Sigma \not\models_{T_s} u(T - X)$ since otherwise there would be some $u(Z) \in \Sigma$ such that $Z \subseteq T - X$. This, however, would mean that $Z \cap X = \emptyset$, which contradicts the hypothesis that $X \in Tr(\mathcal{H})$. It remains to show that for all $H \in X$, $\Sigma \models_{T_s} u((T - X)H)$. Assume the opposite, i.e. there is some $H \in X$ such that $\Sigma \not\models_{T_s} u((T - X)H)$. Then there cannot be any $u(Z) \in \Sigma$ such that $Z \subseteq (T - X)H = T - (X - H)$. Hence, for all $u(Z) \in \Sigma$ we have $Z \cap (X - H) \neq \emptyset$. This, however, contradicts the minimality of $X \in Tr(\mathcal{H})$. We have shown that $a(T - X) \in \Sigma^{-1}$.

We show now that if $a(X) \in \Sigma^{-1}$, then $T - X \in Tr(\mathcal{H})$. From $a(X) \in \Sigma^{-1}$ we conclude that $\Sigma \not\models_{T_s} u(X)$. Hence, for all $u(Z) \in \Sigma((T - X) \cap Z \neq \emptyset)$. From $a(X) \in \Sigma^{-1}$ we know that for all $H \in T - X$, $\Sigma \models_{T_s} u(XH)$. Hence, for all $H \in T - X$ there is some $u(Z) \in \Sigma$ such that $Z \subseteq XH$. Thus, for all $H \in T - X$ there is some $u(Z) \in \Sigma$ such that $(T - XH) \cap Z = \emptyset$. That is, $T - X \in Tr(\mathcal{H})$. □

We have now a complete strategy for computing Armstrong tables. That is, we first compute the set of anti-keys, and then produce rows whose strong agree sets match these anti-keys. The algorithm can also handle the special case where Σ contains the empty key $u(\emptyset)$, saying that each table can have at most one row. This case is dealt with in step $(A0)$. The final step $(A4)$ introduces null marker occurrences in columns that do not belong to the NFS.

Algorithm 2 . (Armstrong table computation)

```
Input:   $(T, \Sigma, nfs(T_s))$ with a set $\Sigma$ of UCs, and NFS $nfs(T_s)$ over $T$;
Output:  Armstrong table $t$ over $T$ for $\Sigma$ and $nfs(T_s)$
Method:  let $c_{H,0}, c_{H,1}, \ldots \in dom(H)$ be distinct
(A0) if $u(\emptyset) \in \Sigma$ then return
         $t := \{r_0\}$ such that $\forall H \in T$ we have $r_0(H) := \begin{cases} c_{H,0} & \text{, if } H \in T_s \\ \texttt{ni} & \text{, else} \end{cases}$;
       else $t = \{r_0\}$ where for all $H \in T$ we have $r_0(H) = \{c_{H,0}\}$ endif;
(A1) compute $\Sigma^{-1}$ using hypergraph transversal methods;
(A2) $i := 1$;
(A3) for all $Y \in \Sigma^{-1}$ do
         $t := t \cup \{r_i\}$ where $r_i(H) := \begin{cases} c_{H,0} & \text{, if } H \in Y \\ c_{H,i} & \text{, if } H \in T_s - Y \\ \texttt{ni} & \text{, else} \end{cases}$;
         $i := i + 1$;
     endfor;
(A4) $total(t) := \{H \in T \mid \forall r \in t(r(H) \neq \texttt{ni})\}$;
     if $total(t) - T_s \neq \emptyset$, then return $t := t \cup \{r_i\}$ where $\forall H \in T$
         $r_i(H) := \begin{cases} \texttt{ni} & \text{, if } H \in total(t) - T_s \\ c_{H,i} & \text{, else} \end{cases}$
     else return $t$ endif;                                                    □
```

Example 6. Let SCHEDULE $= \{CTLR\}$ with SCHEDULE$_s$ $= \{CT\}$. Let Σ consist of the two UCs $u(CT)$ and $u(LTR)$. On input (SCHEDULE$, \Sigma,$ SCHEDULE$_s$), Algorithm 2 would compute the following Armstrong table:

C_ID	Time	L_Name	Room
$c_{H,0}$	$c_{T,0}$	$c_{L,0}$	$c_{R,0}$
$c_{H,0}$	$c_{T,1}$	$c_{L,0}$	$c_{R,0}$
$c_{H,1}$	$c_{T,0}$	$c_{L,0}$	ni
$c_{H,2}$	$c_{T,0}$	ni	$c_{R,0}$

A suitable value substitution yields the sample on the left of Table 2. □

The following result follows directly from Lemma 2 and Theorem 1.

Theorem 3. *On input $(T, \Sigma, nfs(T_s))$, Algorithm 2 computes a table t over T that is Armstrong for Σ and $nfs(T_s)$.* □

4.4 Complexity Considerations

In this subsection, we investigate properties regarding the space and time complexity of the computation of Armstrong tables.

Worst-Case Time-Complexity. The following result follows straight from Theorem 1 and the correctness of Algorithm 2.

Proposition 1. *Let Σ be a set of UCs and $nfs(T_s)$ be some NFS over table schema T. Let t be an Armstrong table for Σ and $nfs(T_s)$. Then $|\Sigma^{-1}| \leq |ag^s(t)| \leq \binom{|t|}{2}$.* □

We recall what we mean by *precisely exponential* [3]. Firstly, it means that there is an algorithm for computing an Armstrong table, given a set Σ of UCs and an NFS $nfs(T_s)$, where the running time of the algorithm is exponential in the number of column headers. Secondly, it means that there is a set Σ of UCs and an NFS $nfs(T_s)$ in which the number of rows in each minimum-sized Armstrong table for Σ and $nfs(T_s)$ is exponential.

Proposition 2. *The complexity of finding an Armstrong table, given a set of UCs and an NFS, is precisely exponential in the number of column headers.* □

Minimum-Sized Armstrong Tables. Despite the general worst-case exponential complexity in the number of column headers, Algorithm 2 is a fairly simple algorithm that is, as we show now, quite conservative in its use of time. Let the size of an Armstrong table be defined as the number of rows that it contains. It is a practical question to ask how many rows a minimum-sized Armstrong table requires. An Armstrong table t for Σ and $nfs(T_s)$ is said to be *minimum-sized* if there is no Armstrong table t' for Σ and $nfs(T_s)$ such that $|t'| < |t|$. That is, for a minimum-sized Armstrong table for Σ and $nfs(T_s)$ there is no Armstrong table for Σ and $nfs(T_s)$ with a smaller number of rows.

Proposition 3. *Let Σ be a set of UCs, $nfs(T_s)$ an NFS over table schema T. Let t be a minimum-sized Armstrong table for Σ and $nfs(T_s)$. Then $\frac{\sqrt{1+8\cdot|\Sigma^{-1}|}}{2} \leq |t| \leq |\Sigma^{-1}|+2$.* □

Note the bounds in Proposition 3. In practice, the number $|\Sigma^{-1}|$ of anti-keys will usually be small. For such typical cases, our Armstrong tables are therefore small as well. The focus on UCs can yield Armstrong tables with a substantially fewer rows than Armstrong tables for more expressive classes such as FDs. The reason is that we do not need to violate any FD not implied by the given set. In practice, this is desirable for the validation of schemata known to be in Boyce-Codd normal form, for example. Such schemata are often the result of Entity-Relationship modeling. Applying the algorithm from [10], designed for UCs and FDs, to our running example yields an Armstrong table with 12 rows. Instead, Algorithm 2, designed for UCs only, produces an Armstrong table with 4 rows. More generally, we can conclude that Algorithm 2 always computes an Armstrong table of reasonably small size.

Corollary 1. *On input $(T, \Sigma, nfs(T_s))$, Algorithm 2 computes an Armstrong table for Σ and $nfs(T_s)$ whose size is at most quadratic in the size of a minimum-sized Armstrong table for Σ and $nfs(T_s)$.* □

Size of Representations. We show that, in general, there is no most concise way of representing the information inherent in a set of UCs and a null-free subschema.

Theorem 4. *There is some set Σ of UCs and an NFS $nfs(T_s)$ such that Σ has size $\mathcal{O}(n)$, and the size of a minimum-sized Armstrong table for Σ and $nfs(T_s)$ is $\mathcal{O}(2^{n/2})$. There is some table schema T, some NFS $nfs(T_s)$ and some set Σ of UCs over T such that there is an Armstrong table for Σ and $nfs(T_s)$ where the number of rows is in $\mathcal{O}(n)$, and the number of minimal UCs implied by Σ in the presence of $nfs(T_s)$ is in $\mathcal{O}(2^n)$.*

Proof. For the first claim let $T = H_1, \ldots, H_{2n}$, $T_s = T$ and $\Sigma = \{u(H_{2i-1}H_{2i}) \mid i = 1, \ldots, n\}$. Then $\Sigma^{-1} = \{a(X_1 \cdots X_n) \mid \forall i = 1, \ldots, n(X_i \in \{H_{2i-1}, H_{2i}\})\}$.

For the second claim let $T = H_1H_1' \cdots H_nH_n'$, $T_s = T$ and $\Sigma = \{u(X_1 \cdots X_n) \mid \forall i = 1, \ldots, n(X_i \in \{H_i, H_i'\})\}$. Then the set of minimal UCs implied by Σ is Σ itself. Since $\Sigma^{-1} = \{a(H_1H_1' \cdots H_{i-1}H_{i-1}'H_{i+1}H_{i+1}' \cdots H_nH_n') \mid i = 1, \ldots, n\}$ there is an Armstrong table for Σ and $nfs(T_s)$ where the number of rows is in $\mathcal{O}(n)$. □

We can see that the representation in form of an Armstrong table can offer tremendous space savings over the representation as a UC set, and vice versa. It seems intuitive to use the representation as Armstrong tables for the discovery of semantically meaningful constraints, and the representation as constraint sets for the discovery of semantically meaningless constraints. This intuition has been confirmed empirically for the class of functional dependencies over relations [13].

5 Mining Keys from SQL Tables

In this section we will establish algorithms for the automated discovery of uniqueness constraints from given SQL tables. Such algorithms have direct applications in schema design, query optimization, and the semantic sampling of databases. In requirements engineering, for example, these algorithms can be utilized to discover semantically meaningful uniqueness constraints from sample data that domain experts provide.

5.1 Mining by Pairwise Comparison of Rows

Our first algorithm gradually inspects all pairs of rows of the given table, and adjusts the set of minimal uniqueness constraints accordingly. Note that the output of Algorithm 5 is $uc(t) = \emptyset$ whenever t contains any duplicate rows.

Algorithm 5 . (Mining of UCs by pairwise row comparison)

Input: table t over T;
Output: set $uc(t)$ of minimal UCs over T satisfied by t
Method:

```
$uc(t) := \{u(\emptyset)\}$;
for all $r_1, r_2 \in t$ such that $r_1 \neq r_2$ do
    for all $u(X) \in uc(t)$ do
        if $X \subseteq ag^s(r_1, r_2)$ then
            $uc(t) := uc(t) - \{u(X)\}$;
            for all $H \in T - X$ where $H \notin ag^s(r_1, r_2)$ do
                $uc(t) := uc(t) \cup \{u(XH)\}$;
            enddo;
        endif;
    enddo;
    while $u(X), u(Y) \in uc(t)$ where $X \subseteq Y$, remove $u(Y)$ from $uc(t)$;
enddo;
```
□

Example 7. Let t, t' be the samples over Schedule from Table 2, respectively. The following table shows the evolution of the minimal UCs by gradually adding a new pair of rows until the entire table has been explored.

rows:	1, 2	1, 3	1, 4	2, 3	2, 4	3, 4
$uc(t)$:	T	TR, TC	TRL, TC	TRL, TC	TRL, TC	TRL, TC
$uc(t')$:	T	TR, TL, TC	TR, TL, TC	TR, TL, TC	TR, TL, TC	TR, TL, TC

The UCs are those explained in the introductory section of this article. □

Theorem 6. *On input (T, t), Algorithm 5 computes the set of minimal UCs satisfied by table t over T in time $\mathcal{O}(|T|^2 \times |t|^2 \times m_t^2)$ where m_t denotes the maximum number of minimal UCs satisfied by any subset $s \subseteq t$.* □

5.2 Mining by Exploration of Hyper-Graph Transversals

Our next algorithm computes transversals for the complements of strong agree sets in the given table. The algorithm is compact and benefits from any progress on the popular problem of computing hyper-graph transversals [8].

Algorithm 7 . (Mining of UCs by exploring hyper-graph transversals)

Input: table t over T;
Output: set $uc(t)$ of minimal UCs over T satisfied by t
Method:

```
$disag^w(t) := \{T - ag^s(r_1, r_2) \mid r_1, r_2 \in t \wedge r_1 \neq r_2\}$;
$nec\text{-}disag^w(t) := \{X \in disag^w(t) \mid \neg\exists Y \in disag^w(t)(Y \subset X)\}$;
$\mathcal{H} := (T, nec\text{-}disag^w(t))$;
$uc(t) := \{u(X) \mid X \in Tr(\mathcal{H})\}$;
```
□

Example 8. Let t, t' be the tables over SCHEDULE from Table 2. The next table shows the steps of applying Algorithm 7 to (SCHEDULE, t) and (SCHEDULE, t'), respectively.

	$ag^s(\cdot)$	$nec\text{-}disag^w(\cdot)$	$u(\cdot)$
t:	CLR, TL, TR, L, R, T	T, CR, CL	$u(CT), u(TLR)$
t':	$CLR, T, \emptyset$	T, CLR	$u(CT), u(LT), u(RT)$

The UCs are those explained in the introductory section of this article. □

The following lemma explains the soundness of Algorithm 7.

Lemma 3. *Let t be a table over table schema T. Then for all $X \subseteq T$, $X \in Tr(T, nec\text{-}disag^w(t))$ if and only if $\models_t u(X)$ and for all $H \in X$, $\not\models_t u(X - H)$.*

Proof. We begin by showing that the two conditions are necessary for every $X \in Tr(T, nec\text{-}disag^w(t))$. First, if t violated $u(X)$, then there were two rows $r_1, r_2 \in t$ such that $r_1 \neq r_2$ and $X \subseteq ag^s(r_1, r_2)$. Hence, $X \cap disag^w(r_1, r_2) = \emptyset$ and there would be some $Y \in nec\text{-}disag^w(t)$ such that $Y \subseteq disag^w(r_1, r_2)$. Thus, $X \cap Y = \emptyset$ which contradicts the hypothesis that $X \in Tr(T, nec\text{-}disag^w(t))$. We conclude that t satisfies $u(X)$. Suppose there was some $H \in X$ such that $\models_t u(X - H)$. Then it would follow that for all $r_1, r_2 \in t$ such that $r_1 \neq r_2$ it held that $(X - H) \cap disag^w(r_1, r_2) \neq \emptyset$. This, however, would violate the minimality of $X \in Tr(T, nec\text{-}disag^w(t))$. Therefore, it holds that for all $H \in X$, $\not\models_t u(X - H)$.

We show now that the two conditions are sufficient for X to be in $Tr(T, nec\text{-}disag^w(t))$. From $\models_t u(X)$ follows that for all $r_1, r_2 \in t$ where $r_1 \neq r_2$ we have $X \cap disag^w(r_1, r_2) \neq \emptyset$. From $\not\models_t u(X - H)$ for all $H \in X$ it follows that for all $H \in X$ there are some $r_1, r_2 \in t$ such that $r_1 \neq r_2$ and $X - H \subseteq ag^s(r_1, r_2)$. The last condition means that $(X - H) \cap disag^w(r_1, r_2) = \emptyset$ holds. Consequently, for all $H \in X$ there is some $Z \in nec\text{-}disag^w(t)$ such that $Z \subseteq disag^w(r_1, r_2)$, and thus $(X - H) \cap Z = \emptyset$. We conclude that $X \in Tr(T, nec\text{-}disag^w(t))$. □

The upper bound in the next result follows from the correctness of Algorithm 7 and known upper bounds for the hyper-graph transversal problem [8].

Theorem 8. *On input (T, t), Algorithm 7 computes the set of minimal UCs satisfied by table t over T in time* $\mathcal{O}\left(|T|^2 \times |t|^2 \times |nec\text{-}disag^w(t)| + \left(\prod_{X \in nec\text{-}disag^w(t)} |X|\right)^2\right)$. □

6 Empirical Measures of Usefulness

It is non-trivial to measure the usefulness of Armstrong tables for the acquisition of meaningful SQL keys. One may conduct a two-phase experiment where design teams are given an application domain and are asked to specify the set of UCs they perceive as meaningful. In the first phase, they are only given a natural language description by domain experts. In the second phase, they are also given an Armstrong table for the set of UCs they perceive currently meaningful.

Whenever new UCs are identified a corresponding Armstrong table can be produced repeatedly. For an experiment or assignment, one may specify a target set Σ_t and possibly a target NFS $nfs(T_s^t)$. One may then measure the quality of the output sets (Σ_i, T_s^i) of the ith phase against the target set (Σ_t, T_s^t), for $i = 1, 2$. If there is an increase in quality, then Armstrong tables are useful indeed. The question remains how to measure the quality of the output sets against the target sets. For this purpose we propose three measures. Let $min(\Sigma)$ denote the UCs in Σ that are minimal. *Soundness* measures which of the (minimal) UCs and headers currently perceived as meaningful are actually meaningful:

$$sound_{\Sigma_t, T_s^t}(\Sigma, T_s) = \frac{|min(\Sigma) \cap \Sigma_t^*| + |T_s \cap T_s^t|}{|min(\Sigma)| + |T_s|}.$$

If $\Sigma = \emptyset$ and $T_s = \emptyset$, we define $sound_{\Sigma_t, T_s^t}(\Sigma, T_s) = 1$. *Completeness* measures which of the actually meaningful (minimal) UCs and headers in NFSs are currently perceived as meaningful:

$$complete_{\Sigma_t, T_s^t}(\Sigma, T_s) = \frac{|\Sigma^* \cap min(\Sigma_t)| + |T_s \cap T_s^t|}{|min(\Sigma_t)| + |T_s^t|}.$$

If $\Sigma_t = \emptyset$ and $T_s^t = \emptyset$, we define $complete_{\Sigma_t, T_s^t}(\Sigma, T_s) = 1$. Finally, *proximity* combines soundness and completeness:

$$prox((\Sigma, T_s), (\Sigma_t, T_s^t)) = \frac{|(min(\Sigma) \cap \Sigma_t^*) \cup (\Sigma^* \cap min(\Sigma_t))| + |T_s \cap T_s^t|}{|min(\Sigma) \cup min(\Sigma_t)| + |T_s \cup T_s^t|}.$$

If $\Sigma = \emptyset = \Sigma_t$ and $T_s = \emptyset = T_s^t$, we define $prox((\Sigma, T_s), (\Sigma_t, T_s^t)) = 1$.

Example 9. Let $\Sigma_t = \{u(CT), u(LT), u(RT)\}$, $\text{Schedule}_s^t = CT$ over table schema Schedule. Let $\Sigma = \{u(CT), u(LRT), u(CLR)\}$ and $T_s = LRT$. Then $sound_{\Sigma_t, T_s^t}(\Sigma, T_s) = 1/2$, $complete_{\Sigma_t, T_s^t}(\Sigma, T_s) = 2/5$, and $prox((\Sigma, T_s), (\Sigma_t, T_s^t)) = 1/3$. □

In database courses one may use Armstrong tables as automated feedback to solutions. Our measures can be applied to automatically mark non-multiple choice questions. This can reduce errors and save time in assessing course work.

7 Conclusion and Future Work

We investigated the data- and schema-driven discovery of SQL keys. We established insights into properties of Armstrong tables. These can increase the discovery of semantically meaningful SQL keys in practice, leading to better schema designs and improved data processing. We established algorithms to automatically discover SQL keys in given SQL tables. These have applications in requirement acquisition, schema design and query optimization. A scenario of how our algorithms can be used together in practice is given in the introduction. We defined measures to assess the difference between sets of SQL keys. These can be applied to validate the usefulness of Armstrong tables and to database education. For the future we plan to implement our results in a design aid, to test our measures in applications, and to address the class of foreign keys.

Acknowledgement. This research is supported by the Marsden fund council from Government funding, administered by the Royal Society of New Zealand.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Atzeni, P., Morfuni, N.: Functional dependencies and constraints on null values in database relations. Information and Control 70(1), 1–31 (1986)
3. Beeri, C., Dowd, M., Fagin, R., Statman, R.: On the structure of Armstrong relations for functional dependencies. J. ACM 31(1), 30–46 (1984)
4. CA Technologies. ERwin Data Modeler - methods guide, p. 86 (2011), https://support.ca.com/cadocs/0/e002961e.pdf
5. Codd, E.F.: A relational model of data for large shared data banks. Commun. ACM 13(6), 377–387 (1970)
6. De Marchi, F., Petit, J.-M.: Semantic sampling of existing databases through informative Armstrong databases. Inf. Syst. 32(3), 446–457 (2007)
7. Demetrovics, J.: On the equivalence of candidate keys with Sperner systems. Acta Cybern. 4, 247–252 (1980)
8. Eiter, T., Gottlob, G.: Identifying the minimal transversals of a hypergraph and related problems. SIAM J. Comput. 24(6), 1278–1304 (1995)
9. Fagin, R.: Armstrong databases. Technical Report RJ3440(40926), IBM Research Laboratory, San Jose, California, USA (1982)
10. Hartmann, S., Kirchberg, M., Link, S.: Design by example for SQL table definitions with functional dependencies. The VLDB Journal (2011), doi:10.1007/s00778-011-0239-5
11. Huhtala, Y., Kärkkäinen, J., Porkka, P., Toivonen, H.: TANE: An efficient algorithm for discovering functional and approximate dependencies. The Computer Journal 42(2), 100–111 (1999)
12. Imielinski, T., Lipski Jr., W.: Incomplete information in relational databases. J. ACM 31(4), 761–791 (1984)
13. Langeveldt, W.-D., Link, S.: Empirical evidence for the usefulness of Armstrong relations in the acquisition of meaningful functional dependencies. Inf. Syst. 35(3), 352–374 (2010)
14. Mannila, H., Räihä, K.-J.: Design by example: An application of Armstrong relations. J. Comput. Syst. Sci. 33(2), 126–141 (1986)
15. Mannila, H., Räihä, K.-J.: Algorithms for inferring functional dependencies from relations. Data Knowl. Eng. 12(1), 83–99 (1994)
16. Sismanis, Y., Brown, P., Haas, P.J., Reinwald, B.: GORDIAN: Efficient and scalable discovery of composite keys. In: VLDB, pp. 691–702 (2006)
17. Zaniolo, C.: Database relations with null values. J. Comput. Syst. Sci. 28(1), 142–166 (1984)

A Framework for Realizing Artifact-Centric Business Processes in Service-Oriented Architecture

Kan Ngamakeur, Sira Yongchareon, and Chengfei Liu

Faculty of Information and Communication Technologies
Swinburne University of Technology, Victoria, Australia
{kngamakeur,syongchareon,cliu}@swin.edu.au

Abstract. Over the past few years, the artifact-centric approach to workflow modeling has been beneficially evidenced for both academic and industrial researches. This approach not only provides a rich insight to key business data and their evolution through business processes, but also allows business and IT stakeholders to have a single unified view of the processes. There are several studies on the modeling and its theoretical aspects; however, the possible realization of this approach in a particular technology is still in its fancy stage. Recently, there exist proposals to achieve such realization by converting from artifact-centric model to activity-centric model that can be implemented on existing workflow management systems. We argue that this approach has several drawbacks as the transformation, which is unidirectional, poses loss of information. In this paper, we propose a framework for the realization of artifact-centric business processes in service-oriented architecture achieving a fully automated mechanism that can realize the artifact-centric model without performing model transformation. A comprehensive discussion and comparison of our framework and other existing works are also presented.

1 Introduction

To meet the challenges of globalization, business processes demand for technologies that can support more efficient and economical way of automation and collaboration. Promisingly, Service-Oriented Architecture (SOA) shows itself as technology enabler that can support such needs. During the recent years, an artifact-centric approach to business process modeling has been introduced as a propitious paradigm that lends itself well to SOA design principle and model-driven architecture (MDA) design concept [1, 3, 5, 7, 12]. This approach has been evidenced in both academic and industrial researches where it not only provides higher level of flexibility of workflow enactment and evolution, but also facilitates the process of business transformation and helps communicating the business intent for consolidating business operations across organizations [1, 2, 3, 4, 5, 8, 9]. In essence, the approach has a central focus on defining key business entities, so called "business artifacts", which are evolved and manipulated within a process. The controlling mechanism that governs the whole process can be implemented by business rules. So far, there have been several studies on the modeling and theoretical aspects of the artifact-centric approach; however, its

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 63–78, 2012.
© Springer-Verlag Berlin Heidelberg 2012

realization and system implementation, especially under SOA and MDA environment, is still in its fancy stage.

One possible and practical approach for realizing the artifact-centric business processes is by transforming an artifact-centric model to a conceptual flow model, which is an activity-centric (control-flow) model. The conceptual flow model is, then, mapped into an executable workflow, e.g., BPEL [5]. The advantage of using this approach is an ease of implementation as workflow technologies and standards based on the traditional model have been developed, e.g., in [11, 12]. In spite of such good point, we argue that this approach has several drawbacks as the transformation, which is unidirectional, poses loss of information. By converting the model, business rules are degraded into control flows; therefore, it is difficult to track and manage the rules based on the converted model. The flexibility of the process is also reduced as business rules are not available to be modified at run-time. Another possible approach is to realize the artifact-centric process model directly without converting the model. This can be considered as more efficient and automatic approach for realizing the model. We claim that the latter approach overcomes the issues of the former approach. In this paper, we propose a framework for the realization of artifact-centric business processes. The framework consists of artifact-centric workflow model and a mechanism that can automatically realize and execute the model under the service-oriented environment. We also provide detailed discussions on technical issues and challenges of our realization framework as well as the comparison with existing activity-centric workflow systems.

The remainder of this paper is organized as follows. Section 2 presents an artifact-centric approach to process modeling. Section 3 discusses the realization framework for artifact-centric business processes. Section 4 shows implementation and evaluation of our framework. Section 5 discusses and reviews the related works. Finally, the conclusion and future work are given in Section 6.

2 Artifact-Centric Approach to Business Process Modeling

In this section, we introduce an example of business processes to illustrate that we can identify business artifacts and use them to construct an artifact-centric business process in order to use it for analysing and capturing the requirement of our prototype system. In the artifact-centric approach, a business process can be constructed using business artifacts. An artifact stores its business relevant information and its lifecycle. The state transition of artifacts is achieved by a service and is controlled by a set of business rules. Our example of business process is adapted from a simple online ordering process. The process starts when a customer places an order including billing information through a web site. Then the order is sent to a manufacturing factory where the ordered product is assembled, tested and packaged. Finally, the product is shipped to the customer. After we examine this process, several business artifacts are identified. Fig. 1 shows *data model* and *lifecycle model* of key business artifacts involved with this business process. For each artifact, the data model represents its data attributes, while the lifecycle model represents its state transition of the artifact.

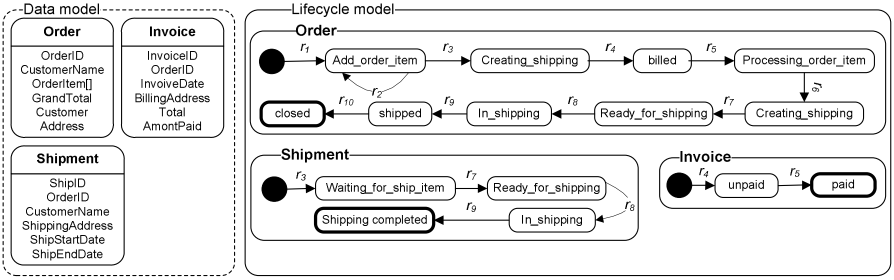


Fig. 1. Artifacts and their lifecycles in product ordering processes

We can see that this process consists of three classes of artifacts: *invoice*, *shipment* and *order*. Apart from the artifacts, in the lifecycle model we can see two components that are essential for constructing a complete business process – those are *services* (a.k.a. tasks) and *business rules*. A service is used to make change on artifacts. An association between services and business artifact(s) is specified by using business rules as to describe on what condition such service is performed on the artifact(s). More details are described in Section 3.2. Based on initial concept of artifact-centric business processes, we analyzed the problem domain to understand basic requirements that needed to be addressed in our framework. Here, we summarized our requirements into three points listed in following paragraphs.

– **A Formal Process Definition of Artifact-Centric Business Process (ACP)**

In artifact-centric approach, the conceptual model of an artifact-centric business process is defined in a declarative manner. The conceptual model provides a high level specification of a business process execution. Normally, it is used to communicate business intents between stakeholders but it can't be executed by a computer system. In order to realize the conceptual model of an artifact-centric business process, we need to develop a process definition that contains all concrete details required by a process execution.

– **A Process Deployment and Execution**

The process deployment has to be developed in such a way that it can parse the model definition, map parsed data to predefined classes, and deploy a process in a web service environment. When a client invokes the deployed process, the process and other related (e.g., artifacts, services, rules) instances need to be created. The concept of executing and managing these instances for artifact-centric business processes are new and relatively challenging. This is because the core constructs of the artifact-centric process model differ from those of the traditional activity-centric model.

– **Business Rule Definition and Evaluation**

In the traditional approach, business logics are defined explicitly using control flows and activities. In contrast, in artifact-centric approach, we use business rules to define an association between artifacts and services. Each rule describes which service is invoked and which artifact(s) is changed under what conditions. This requires an investigation of how rules can be defined in the most expressive and effective manner. In the implementation, the integration of a suitable rule engine to our system to handle the artifact-centric process execution is also challenging.

3 ACP Realization Framework

In this section, we illustrate our framework for automated realization of artifact-centric business processes. The detailed technical discussion on proposed system architecture is also presented. It is quite easy to comprehend the artifact-centric business process model at the conceptual level from our motivating example since it was designed to incorporate information and behaviour aspects of a business process. As a result, we can convey and communicate business intent among a variety of stakeholders. As already introduced, two approaches are observed. The first approach is to convert the artifact-centric model into a conceptual model in a procedural manner. The good side is that the artifact model can be easily to be implemented using traditional workflow technologies. Its drawback is that the flexibility of the artifact model and data may lose in the model conversion. On the contrary, our approach directly realizes the artifact-centric process model. Here, we propose an ACP realization framework based on the direct approach, and it is illustrated in Fig. 2.

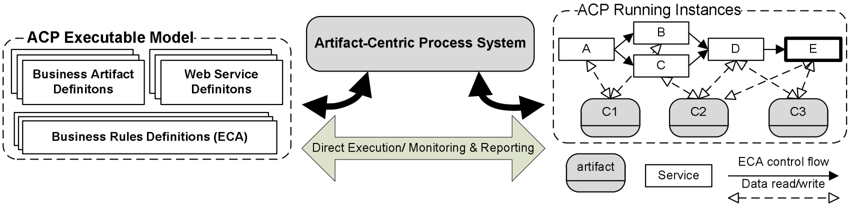

ACP Executable Model
Business Artifact Definitons
Web Service Definitons
Business Rules Definitions (ECA)
Artifact-Centric Process System
Direct Execution/ Monitoring & Reporting
ACP Running Instances
A
B
C
D
E
C1
C2
C3
artifact
Service
ECA control flow
Data read/write

Fig. 2. ACP realization framework

In our framework, we aim at fully automated realization from ACP definition to its execution. This framework does not require additional model transformation (from ACP model to workflow executable model) nor require backward mapping mechanism to validate the running instances with the original ACP model. The framework contains only ACP executable model and the automated realization mechanism that can directly execute such process definitions. In the task-based model, data is defined separately at later time (in most cases after the task has already been defined). While in run-time, workflow systems do not realize the relationship between the current stage of task execution and the state of the data or artifacts that being manipulated. This poses a technical problem when attempting to discover the correspondence and to track run-time instances of those running artifacts directly in the ACP model. In our approach, instances of process, services, and artifacts being manipulated can be directly reported regarding their ACP model. Monitoring a progress of a particular business process can be efficiently achieved at both artifact and process levels. We can see that such direct monitoring and reporting are more efficient as an additional reverse mapping from instances of activity-centric model to artifact-centric model is not needed.

3.1 Artifact-Centric Business Process Model

Here, we introduce an *artifact-centric business process model* (*ACP model*) that has been proposed in our previous work [9, 10]. Our ACP model consists of sets of *artifact classes*, *services*, and *business rules*. An artifact, which is a key business entity involved in business processes, contains its relevant attributes and many finite processing states. Let $Z = \{C_1,\ C_2\ ...,\ C_x\}$ be a finite set of artifact classes that are used in a particular process. Each *artifact class* $C_i \in Z$ is defined as a tuple (A, $\boldsymbol{s^{init}}$, S, $\boldsymbol{S^f}$) where set $A = \{a_1,\ a_2,\ \ldots,\ a_y\}$, and each $a_j \in A$ is a name-value pair attribute; set $S = \{s_1,\ s_2, \ldots,\ s_z\}$ contains the possible states of the instances of class C_i; $\boldsymbol{s^{init}}$ is the *initial* state, and $\boldsymbol{S^f} \subseteq S$ is a set of its *final* states. A *service* is a task that is used to perform read/write operations on some artifact(s), and it is denoted as $v(C_1,\ C_2,\ ..,\ C_y)$ where $C_1,\ C_2,\ ..,\ C_y$ are artifacts that are read/updated by service v. A *business rule* is used to associate service(s) with artifact(s). It is defined in a *Condition-Action* style to describe on what *pre-condition* a particular service is executed, and on what *post-condition* after performing such service must satisfy. A *business rule*, denoted as r, is a tuple (λ, β, v) where λ and β are a *pre-condition* and *post-condition*, respectively; v is a service that performs read/update operations on the attributes and the processing states of some artifacts in schema Z. We restrict both pre- and post-conditions to be expressed by a conjunctive normal form. This form can contain two types of proposition over schema Z: (1) *state proposition* (by *instate* predicate) and (2) *attribute proposition* (by *defined* predicate and scalar comparison operators). We write *defined*(C, a) if attribute $a \in C.A$ of artifact of class C has a value; and *instate*(C, s) if state $s \in C.S$ of artifact of class C is active. Initially, *instate*(C, $\boldsymbol{s^{init}}$) implies $\forall x \in C.A$, $\neg$*defined*(C, x). A complete set of business rules defined for a particular process model specifies the control logic (named ECA flow) of the whole process from the beginning to the termination of the process. Table 1 shows an example subset of business rules that are used in our product ordering process.

Table 1. Example of business rules

r1: *Customer requests to make an order O*	
Pre-condition	*instate(O,init) ∧ defined(O,OrderID) ∧ defined(O.CustomerName) ^ defined(O.CustomerAddress)*
Service	*createOrder(O)*
Post-condition	*instate(O,Add_OrderItem) ^ defined(O.OrderID) ^ defined(O.CustomerName) ^ defined(O.CustomerAddress)*
r2: Create Shipment S for an order O	
Pre-condition	*instate(O,Add_Order_Item) ^ instate(S,Init) ^ defined(O. GrandTotal) ^ O.GrandTotal>0 ^ defined(S.ShipID) ^ defined(S.OrderID) ^ defined(S.ShippingAddress)*
Service	*createShipping(S,O)*
Post-condition	*instate(O,Create_Shipping) ^ instate(S,waiting_for_Ship_Item) ^ defined(S.CustomerName) ^ defined(S.ShippingAddress) ^ defined(S.ShipID) ^ defined(S.OrderID)*
r3: Create Invoice I for an order O	
Pre-condition	*instate(I,Init) ^ instate(O,Creating_Shipping) ^ defined(I.InvoiceID) ^ defined(I.OrderID) ^ defined(I.BillingAddress) ^ defined(I.InvoiceDate) ^ defined(I.Total) ^ I.Total= O.GrandTotal*
Service	*createInvoice(I,O)*
Post-condition	*instate(V,Unpaid) ^ instate(O,Billed)*

3.2 ACP Executable Model

Now, we propose to use a serializable and executable version of the ACP model based on the ACP Model definitions described in Section 3.1. Our artifact-centric executable process model is defined by using XML, and it consists of three definitions: *artifact definition*, *business rule definition*, and *service definition*, as shown in Fig. 3. It contains implementation details required by a system to execute a particular business process and they are used for creating running instances.

```
<artifacts>
    <artifact name = "Order">
        <attributes>
            <attribute name = "orderID" structure = "pair" type = "String" />
        </attributes>
        <states>
            <state name = "start" type = "init" />
            <state name = "open_for_item" />
            <state name = "end" type = "end" />
        </states>
    </artifact>
</artifacts>
```

```
<services>
    <service name = "crteateOrderService"
      inputMessage = "createOrderRequest"
      outputMessage ="createOrderRespone"
      operation ="createOrder"
      port ="createOrderSOAP"
      location = "http://localhost:8080/axis2/services/createOrder?wsdl"
      namespace = "www.swin.edu.au" />
</services>
```

```
<businessrules>
    <rule name = "create_order">
        <onEvent type = "inputMessage"/>
        <precon>
            <and>
               <atom type ="state" artifact="Order" id="Order1" value = "init" />
               <atom type="attribute" artifact="Order" id="Order1" attribute="item_quantity" op="==" value="0"/>
              <atom type="input" attribute="a" op=">" value="1"/>
            </and>
        </precon>
        <do>
            <invoke type = "internal" operation = "createOrder" service = "createOrderService" >
               <mapping>
                   ............

               </mapping>
                <transitions>
                   <transition artifact = "order" id = "order1" fromState = "init" toState = "open_for_item" / >
                </transitions>
            </invoke>
        </do>
    </rule>
</businessrules>
```

Fig. 3. Artifact, Service, and Business rule definitions

- **Artifact Definition** composes of a set of attributes and states. An attribute definition provides details of business data (<name>, <type>, and <structure>) that can be stored in each attribute of a particular artifact, such as attribute Name, data type and data Structure. A state definition provides details of each state (<name>, <type>) in a particular artifact life cycle, such as name of state, initial state and final state.
- **Business Rule Definition** is used to define an ECA-like rule description. This rule consists of event, precondition and action (<onEvent>, <precon>, and <do>). Element <onEvent> provides details of which event can trigger a particular rule. Element <preCon> is a condition that needs to be satisfied in order to take a further action. Element <do> is a task or service that needs to be invoked. Element <invoke> provides service name and operation name of a designated web service. Moreover, Mapping rules and transition rules (<map>, <transition>) are defined in this part as well. Mapping rule provides details of data mapping between artifact and message. The transition rule is used to control state transition for each artifact involving in a step of process execution.
- **Service Definition** defines concrete details of a web service. This definition provides information that is necessary for a service invocation, such as service name, operation, WSDL location, and port. In this paper, we consider only inputs and outputs of a web service not including its behaviours.

Due to the fact that the current web service technologies do not support an artifact as an input of a web service. To address this issue, an internal data mapping mechanism is required in order to correlate the passing messages (input/output) and their corresponding data attributes of artifacts. We use a mapping rule to map data between artifact and SOAP message; therefore we introduce mapping rules, as shown in Fig. 4 and Fig. 5 into our framework. We consider that there are two types of data mapping in our framework, which are mapping from message to artifacts and from artifacts to message. We include mapping type to indicate a direction of mapping. The rule is also included details of a source (<from>) and a destination (<to>) for mapping between artifact and message. These two elements contain information that helps the system to locate corresponding artifact's attribute or message's part to be map when a web service is invoked.

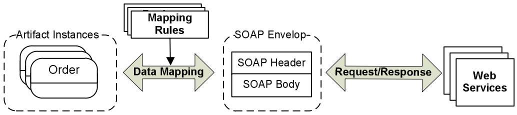


Fig. 4. Data mapping between SOAP Message and Artifacts

```
<map type = "MessageToArtifact">
    <copy>
        <from message = "calTotalMessage" part = "customerID"/>
        <to artifact = "order" attribute="orderID"/>
    </copy>
     <copy>
        <from message = "calTotalMessage" part = "numberOfItem"/>
        <to artifact = "order" attribute="quantity"/>
     </copy>
</map>
```

```
<map type = "ArtifactToMessage">
    <copy>
        <from artifact = "order" attribute="orderID"/>
        <to message = "calTotalMessage" part = "customerID"/>
    </copy>
     <copy>
        <from artifact = "order" attribute="quantity"/>
        <to message = "calTotalMessage" part = "numberOfItem"/>
     </copy>
</map>
```

Fig. 5. Example of mapping rules

3.3 Run-Time ACP Instances

During a process execution, we need to keep track the status of a running process. This allows the system administrator to inspect the status during a runtime and after a completion of a process execution. We classify instances of ACP into four following types where each of which corresponds to individual component of ACP model.

- *Process instance* – When the process is enacted then the system initially creates process instance. Once a process instance is created, it will be given its name corresponding to executed business process and will be given an identifier key. Process instance acts as a container to store other running instances, which are artifact instance, rule instance and service instance.
- *Artifact instance* – In a process, an instance of particular artifact class can be created at the time the process is initialized or after service invocation (that performs a creation of artifact). Newly created artifact instance will be populated with artifact definition data and will be given an artifact identifier key. This instance serves a purpose of storing information including business data and lifecycle during each step of business process execution. Thus, it is a key to indicate progress of a running process.

- *Service instance* – Service is instantiated when it is invoked (as defined by the action in a business rule). It not only stores service invocation information defined in a service definition but it also captures input/output message data as well as timestamp.
- *Rule instance* – An instance of business rule is created when the rule is triggered by event and its pre-condition holds. A rule instance provides information regarding decision making. By inspecting this instance, we will know which rule is fired, what time rule is fired, and what data triggered rule firing.

Based on the above types of instances of ACP, we can gather the complete execution traces by recording every instance type on the log records. These records of a particular process permit the real-time (direct) monitoring of the process and its components without the reverse mapping, which is required for the existing model transformation realization approach, i.e., covert ACP model to task-based model and run it on existing workflow system. It is worthwhile mentioning that with our framework, business rules can be modified/removed/added at run-time while still able to reflect its process model. At run-time, we allow our ACP system to keep different versions of business rule for a particular process model by storing the mapping of every version and its original version. This feature enhances the system ability to be able to track/monitor different versions of (process and rule) instances of the same process. We also claim this feature to be one of the advantages of our realization framework compared with the existing approach.

4 Implementation and Evaluation

4.1 ACP System Architecture and Its Components

In this section we show our proposed architecture of the artifact-centric process system (*ACP System*), as illustrated in Fig. 6. This system architecture ensures that we can address those requirements from the previous section. We adopted the concept of the event driven architecture and service-oriented architecture.

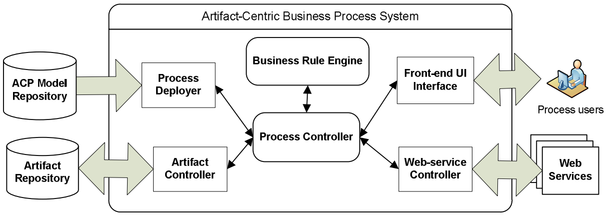


Fig. 6. ACP System

Here, we describe each ACP system's component in more details.

– *Process Deployer* is used to deploy ACP Model definition file. The definition file will be parsed to generate running instances of a particular business process.

- *Business Rule Engine* provides a rule evaluating functionality. For any change in a running process, a rule engine will evaluate an instance in order to determine the next possible action that will be undertaken.
- *Process Controller* is used to manage instances of a process based on rule engine's command. Once the process controller receives a command from rule engine to start a new process execution, it will use a process factory to create running process instance. The factory will identify the corresponding instance and create it. The created process instance will be given an id for identification purpose. The process controller can issue a command to an artifact controller to update artifact instances or to web-service controller to invoke web services. For any changes in a running process, the process controller will consult the rule engine to perform the next possible action.
- *Web Service Controller* is used to invoke web services to process artifact data. To invoke a web service, the controller creates a soap message corresponding to message definition in WSDL. The artifact data is mapped to message data using a mapping rule. Finally, the request message is sent to a designated web service. Once a response message is returned, the service controller processes the message, and returned message data is mapped to corresponding artifact attribute.
- *Artifact Controller* is used to manage and update artifact (which is stored in external repository). After the service controller receives a response message from a web service, a data mapper will extract the message. Then the artifact controller uses such data to update corresponding artifacts.
- *Front-end UI Interface,* proposed in our previous work [10], is used to manage web-based interactions between ACP system and users, which includes automatic generation of web pages and receiving/responding via web form interfaces.

4.2 Run-Time Execution

Here, we discuss operations of our ACP system in more details including instantiation of a running instance, operation of rule engine and how each component works together to coordinate a process execution.

- **Creation of a Running Instance**

The ACP system has the component so called process factory to create running instances from an ACP definition. Once a process execution has started, the process controller will call a process factory method to create a process instance and also other running instances. The factory will identify a correct deployed process and instantiate corresponding instances. During instantiation of running instances, implantation data stored in the definition file will be parsed and mapped to corresponding instances. Upon receiving a process instance, the process controller will register the process instance in order to be able to keep track processes that are currently running.

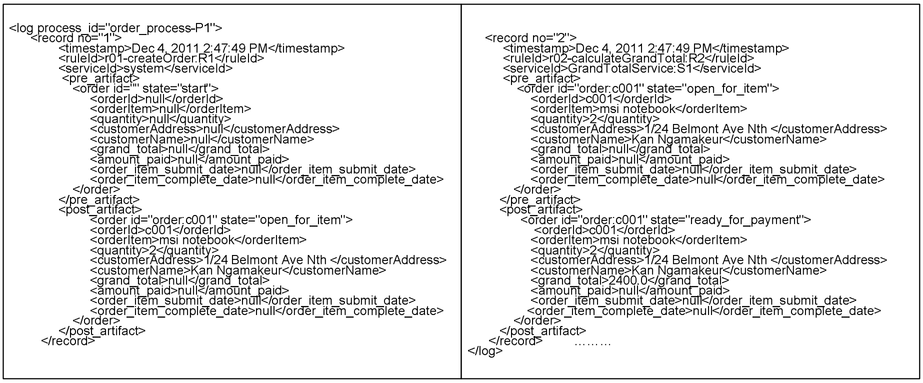
- **Integration of Rule Engine**

In Artifact-centric business process, business rules are main mechanism to control interaction between artifacts and services. The rule engine is integrated into our system to provide functionality for evaluating business rules. The rule engine will be activated once it receives an internal event generated by the process controller. During the activation, a process instance will be feed as an input of a rule engine. A process's data is validated against a set of conditions. If conditions are satisfied, an action will be undertaken to make changes to artifacts. A rule instance that keeps track of rule execution is also created in the process as well. In our current prototype, we have integrated Drools rule engine [21] since the engine provides very efficient ways of evaluating business rules. The rule format is also easy to comprehend and can be written in xml format and drool format. Drools engine conforms to JSR94 standard and provide a set of APIs that allows us to integrate it to our system.

– **Coordination of Running Instances**

In order to coordinate running instances, we address this issue using process controller, artifact controller and service controller. Here, we will use our motivation example to describe how these controllers work together to coordinate all instances that are created during a process execution. Once the ACP system receives a request from a user to start an ordering process, the rule engine will evaluate the request. If preconditions of rule *r1* are satisfied, the rule engine will issue a command to the process controller to start a process execution of an ordering process. The rule instance of rule *r1* is also created at this point as well. The process controller invokes the process factory. The factory identifies the deployed process and instantiate a process instance for an *ordering* process. After receiving a corresponding process instance, the process controller will initialize a unique process id and register the process instance to a list of running process instance. Once a process instance is registered, an instance of rule *r1* is added to a list of rule instances. The artifact instance of an *order* artifact is also created by the factory and added to a list of artifact instances. The artifact instance is given a unique id that is used for identification purpose and its state is set to *start*. Next, the process controller orders a web-service controller to invoke the *createOrder* service. The service instance of *createOrder* service is instantiated and added to a list of service instances. The service controller will communicate with the artifact controller to retrieve data from corresponding artifacts. The *createOrder* service is then invoked. Once a response is returned, a service controller will send message data to an artifact controller to update the artifacts. The unique artifact ensures that correct artifact instance is being updated. A new artifact instance will be generated if necessary during this step as well. Mapping rule defined in rule *r1* controls the data mapping between service input-output and artifact instance. Once finishing data updating, the artifact controller will update a state of an *order* artifact from *start* to *open_for_item*. After completion of artifact updating, the artifact controller sends a signal to the process controller. The process controller will generate *Artifact_change* event to trigger rule engine to continue a process until *ordering* process is completed.

As we can see that each step of a process execution, the ACP system creates artifact instances, service instances and rule instances (cf. Section 3.3). These instances can be used to monitor a process since they contain overall information. To prove our concepts, we developed a prototype of ACP system and generated a test case based on the motivating example. After the prototype executed a test case, it is able to process data from running instances and generate a log file. In a log file, we can see detailed information for each step of a particular business process execution. To generate this log file, the system need to capture data from rule instances, artifact instances and service instance at run-time. A rule instance contains identifier keys that belong to involving service instance and artifact instance. These keys help define a relationship between rule instance and the other instance in each step of a process execution. This enables our system to be able to generate a record for each step during run-time. As shown in Fig. 7, Pre and Post-artifact are also recorded in a log file to show progress of each artifact from initial state to final state. The system records these data before and after service invocation. We can also use these pre and post artifact data to help facilitate process provenance if it is necessary. Therefore, this is a solid proof to illustrate an advantage of artifact-centric business process regarding to monitoring and reporting.



```
<log process_id="order_process-P1">
   <record no="1">
         <timestamp>Dec 4, 2011 2:47:49 PM</timestamp>
         <ruleId>r01-createOrder:R1</ruleId>
         <serviceId>system</serviceId>
          <pre_artifact>
            <order id="" state="start">
                <orderId>null</orderId>
                <orderItem>null</orderItem>
                <quantity>null</quantity>
                <customerAddress>null</customerAddress>
                <customerName>null</customerName>
                <grand_total>null</grand_total>
                <amount_paid>null</amount_paid>
                <order_item_submit_date>null</order_item_submit_date>
                <order_item_complete_date>null</order_item_complete_date>
            </order>
         </pre_artifact>
         <post_artifact>
                <order id="order:c001" state="open_for_item">
                <orderId>c001</orderId>
                <orderItem>msi notebook</orderItem>
                <quantity>2</quantity>
                <customerAddress>1/24 Belmont Ave Nth </customerAddress>
                <customerName>Kan Ngamakeur</customerName>
                <grand_total>null</grand_total>
                <amount_paid>null</amount_paid>
                <order_item_submit_date>null</order_item_submit_date>
                <order_item_complete_date>null</order_item_complete_date>
            </order>
         </post_artifact>
      </record>
```

```
     <record no="2">
         <timestamp>Dec 4, 2011 2:47:49 PM</timestamp>
         <ruleId>r02-calculateGrandTotal:R2</ruleId>
         <serviceId>GrandTotalService:S1</serviceId>
         <pre_artifact>
            <order id="order:c001" state="open_for_item">
               <orderId>c001</orderId>
               <orderItem>msi notebook</orderItem>
               <quantity>2</quantity>
               <customerAddress>1/24 Belmont Ave Nth </customerAddress>
               <customerName>Kan Ngamakeur</customerName>
               <grand_total>null</grand_total>
               <amount_paid>null</amount_paid>
               <order_item_submit_date>null</order_item_submit_date>
               <order_item_complete_date>null</order_item_complete_date>
           </order>
        </pre_artifact>
        <post_artifact>
             <order id="order:c001" state="ready_for_payment">
                <orderId>c001</orderId>
               <orderItem>msi notebook</orderItem>
               <quantity>2</quantity>
               <customerAddress>1/24 Belmont Ave Nth </customerAddress>
               <customerName>Kan Ngamakeur</customerName>
               <grand_total>2400.0</grand_total>
               <amount_paid>null</amount_paid>
               <order_item_submit_date>null</order_item_submit_date>
              <order_item_complete_date>null</order_item_complete_date>
           </order>
        </post_artifact>
      </record>      ………
</log>
```

Fig. 7. Log record of a test case based on the motivating example

4.3 Technical Evaluation

In this section, based on the result of our implementation prototype, we discuss on the technical evaluation of our ACP system as well as a detailed comparison between two realizations approaches. After a prototype of ACP system is completed, we have simulated test cases based on our online ordering process. The result shows that our framework can address our requirements. The developed system is able to manage running instances created during process execution. Each running instance, e.g. service instance, stores process execution data and can be used for purpose of monitoring and reporting as shown in Fig. 7. Log record of a test case based on the motivating example. A business rule engine is proved to be able to work solely to provide decision making that affects on running processes. In our current prototype, we centralize all decision making process into a single rule engine. Thus, it simplifies rule management. However, this may raise performance issue of process execution if there are thousands of business rule to be evaluated by a rule engine. Non-deterministic is also an issue since a rule engine fires rules simultaneously. However, their ordering is non-deterministic. Thus, sequence of process execution may be different even with the same business process. A task for evaluating reachability of running processes is needed to address this issue. In our implementation, we assume that there is no issue regarding to non-deterministic. Since business process models are defined implicitly in artifact-centric approach, this would be another issue for a process modeler. An artifact model doesn't have any explicit control flows as in the traditional process model so this is not an easy task for the process modeler. Thus, an intuitive process designing tool needs to be further developed. As we know that there is another way to implement artifact-centric processes, we compare our system to this existing artifact system implemented using the model transformation approach described in papers [11, 5].

– **Realization Approach**
In our implementation, we used our direct approach to realize an ACP model whereas the opposite approach proposed in [11] attempts the conversion from an artifact-centric process model to a procedural model, e.g. BPEL. We consider that logical information of artifact-centric process model can be lost during the model conversion process since some logical information which defined in a declarative manner, e.g. business rules, is spilt and mapped into several control flows in a procedural model. Moreover, this conversion task is quite cumbersome, error-prone and time consuming. Our approach ensures that there is no loss of data during transformation of the conceptual model since logical information of the conceptual model can be mapped faithfully to the proposed executable model. Without any model conversion, this approach uses less time and reduces chance of making mistake. Thus, direct approach is considered to be more appropriate way to realize the artifact model compared to model conversion approach.

– **Flexibility and Changes Management**
Flexibility is strength of a process model in declarative style. A conceptual model of artifact-centric business process gains this advantage as well since it is specified in the same style. Direct approach that we used to realize the conceptual model guarantees that the executable model inherits flexibility from its conceptual model, whereas the other approach does not since tasks are locked up by control flows. As a result, flexibility is well supported for design-time and run-time for our approach, whereas the other approach partially supports flexibility at run-time as it depends on the functionality offered by a particular workflow system. Thus, Changes can be made directly on the implementation level in our direct realization approach. In contrast, changes have to made at design-time and then convert to the implementation if an artifact model realized in a procedural workflow.

– **Monitoring and Reporting**
As opposed to traditional approach for process modeling, an artifact-centric process model focuses on business artifacts as its first class citizen to model a particular business process. Each business artifact contains business-relevant data and its life cycle. Artifact data and life cycle of each artifact reflect progress of a particular process toward a business goal. Thus, business process monitoring and tracking can be done by inspecting artifacts. Our approach provides a feature of direct and consistent monitoring and reporting at both model and instance level since both data and life cycle are combined at model level and instance level. Our implementation illustrates that a particular process can be monitored by directly inspecting running instances at run-time without any technique involving data gathering and processing. Whereas, the other approach needs a sophisticated mechanism which may include retransformation from the implementation specification back to its model specification and backward mapping for some data to its model to gather and process all process information to provide monitoring and reporting functionality.

– **Verification and Conformance Checking**
Verification and conformance checking is very essential task for both traditional approach and artifact-centric approach for modeling business process to ensure validity of developed model so that it can be realized on an automated system to support decision making for a particular business process. Since we used direct approach to realize artifact-centric business process model, single model verification for both design-time and run-time can be achieved because an implementation level reflects its conceptual level. Thus, conformance checking can be achieved directly, whereas the other realization approach needs to have separate verification on both Artifact model and procedural model. Run-time verification does not reflect the base artifact model because of the conversion. Therefore, conformance checking needs some additional procedures.

– **Standards and Technologies Support**
Although our approach has several advantages, there are some drawbacks regarding to standards and technologies supports. Artifact-centric model realized on traditional workflow benefits from current industry-wide standards and technologies, e.g. OASIS, OMG, W3C and etc. Thus, an implementation of this realization approach is much easier and faster than our approach. Moreover, interoperability and execution in distributed environment are well supported when the artifact model is realized on traditional task-based workflow system. Currently, the developed prototype system only supports execution of an artifact-centric business process model in local environment and need further extension to handle distributed executions.

5 Related Work and Discussion

The notion of a business artifact was originated in [1] where business operational model can be constructed using a collection of lifecycles of all artifacts and their interaction. The operational model based on business artifacts provides the benefits that are flexibility of the representation, ability for analyzing changes, and ability for managing application. Moreover, Rong et al [2] improved the idea of the business operational model by introducing nine operational patterns for constructing the model and the method for verification the model. The concept of business artifact was further adopted in [5] as a business process model can be constructed using four core constructs that are artifacts, artifact lifecycles, services, and association. To realize an artifact-centered model, this paper presented a three-layer framework. The artifact-centric business process model considered as a logical specification sits on the top level. Then, it is converted to a conceptual flow that captures an essence of the top-level model in a procedural manner. Finally, it is mapped into an operational workflow for automation. Gerede and Su [3] focused on the middle layer of the framework, a conceptual flow, as it provides a separation between the logical specification and the physical execution; hence changes can be made freely to the implementation level as long as the logical specification remains unchanged. Therefore, the conceptual flow needs to be verified and optimized to ensure its correctness and performance respectively. This paper presented verification and optimization techniques for a conceptual flow.

There were other works that extend the artifact-centric approach. Yongchareon and Liu [9] introduced a process view framework for artifact-centric business processes followed by the extended version for modeling inter-organizational processes [22].

An (public/private) artifact-centric process view can be used to support participated organizations to have their own freedom to model and implement their own parts of the process while preserving global correctness of the collaboration. Narendra et al. [12] tried to address flexibility and monitoring issues of the service composition using business artifacts, and their lifecycles. The concept of context-based artifact was introduced in this paper. The contexts are not only used to keep track of changes that make on all artifacts but also used in the coordination between artifacts and web services. To support inter-organization, the artifact-centric hubs were proposed by Hull et al. [6]. The hubs provide a centralized, computerized rendezvous point, where stakeholders can access data of the common interest and check the current status of an aggregate process. The framework also incorporates access control mechanisms to cope security issues. Instead of the centralized hubs, Lohmann and Wolf [7] proposed the use of artifacts in a choreography setting. In particular, the artifact-centric business process models were enhanced with the concept of agents and locations. By defining precisely and clearly who is accessing an artifact and where the artifact is located, an interaction model that acts as a contract between the agents can be generated automatically. Liu el al [23] proposed an approach to performance monitoring based on Artifact-centric business. The first step is to create monitoring context skeletons from business-artifact definition and from user inputs. The second step is to derive the executable monitoring models from the monitoring context skeletons. This work may be able to apply to improve our prototype in the future.

As was indicated in [4] by Hull, the implementation of artifact-centric business process is considered as one area of research challenges needed. An artifact process model can be converted to a conceptual flow. Then, it is mapped into an executable workflow. This first approach was adopted in [11]. This paper introduced the conceptual flow, namely ArtiFlow, and showed how ArtiFlow can be mapped to BPEL. Cohn and Hull [8] illustrated that IBM has used BELA tool to map an artifact-centric process model into a workflow that runs on IBM's WebSphere Process Server. In this research, we use a different approach compared with Artiflow. Our realization approach is to generate the executable model from the logical specification of an artifact-centric model based on [9, 10] without any transformation of the model. We also develop our prototype to execute our proposed executable model where the system uses business rules to control each state of process execution. In Siena [15], users can model business artifacts and process as an XML documents in order to create a composite web application. Then, the application is deployed and executed on an execution engine. However, there is no use of business rules. Moreover, processes are still executed in a procedural manner.

6 Conclusion and Future Work

In this paper, we propose a new framework for realizing artifact-centric business processes. Especially, we showed how an artifact-centric process model can be realized in our system. Apart from the proposed system for the realization of ACP model, we also provided a detailed discussion on the advantages and disadvantages of our approach. Our future work will extend our system to support interoperability.

Acknowledgement. This research was partly supported by the Australian Research Council Linkage Project LP0990393.

References

1. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Syst. J. 42(3), 428–445 (2003)
2. Liu, R., Bhattacharya, K., Wu, F.Y.: Modeling Business Contexture and Behavior Using Business Artifacts. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 324–339. Springer, Heidelberg (2007)
3. Gerede, C.E., Su, J.: Specification and Verification of Artifact Behaviors in Business Process Models. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 181–192. Springer, Heidelberg (2007)
4. Hull, R.: Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008)
5. Bhattacharya, K., Hull, R., Su, J.: A data-centric design methodology for business processes. In: Handbook of Research on Business Process Modeling (2009)
6. Hull, R., Narendra, N.C., Nigam, A.: Facilitating Workflow Interoperation Using Artifact-Centric Hubs. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 1–18. Springer, Heidelberg (2009)
7. Lohmann, N., Wolf, K.: Artifact-Centric Choreographies. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 32–46. Springer, Heidelberg (2010)
8. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Data Engineering Bulletin 32(3), 3–9 (2009)
9. Yongchareon, S., Liu, C.: A Process View Framework for Artifact-Centric Business Processes. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010, Part I. LNCS, vol. 6426, pp. 26–43. Springer, Heidelberg (2010)
10. Yongchareon, S., Liu, C., Zhao, X., Xu, J.: An Artifact-Centric Approach to Generating Web-Based Business Process Driven User Interfaces. In: Chen, L., Triantafillou, P., Suel, T. (eds.) WISE 2010. LNCS, vol. 6488, pp. 419–427. Springer, Heidelberg (2010)
11. Liu, G., Liu, X., Qin, H., Su, J., Yan, Z., Zhang, L.: Automated Realization of Business Workflow Specification. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 96–108. Springer, Heidelberg (2010)
12. Narendra, N.C., Badr, Y., Thiran, P., Maamar, Z.: Towards a Unified Approach for Business Process Modeling Using Context-based Artifacts and Web Services. In: IEEE SCC 2009, pp. 332–339 (2009)
13. Hull, R., Kumar, B., Lieuwen, D., Patel-Schneider, P.F., Sahuguet, A., Varadarajan, S., Vyas, A.: Everything Personal, not Just Business: improving user Experience through Rule-Based Service Customization. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 149–164. Springer, Heidelberg (2003)
14. Hull, R., Kumar, B., Lieuwen, D.F., Patel-Schneider, P.F., Sahuguet, A., Varadarajan, S., Vyas, A.: Enabling context-aware and privacy-conscious user data sharing. In: IEEE Intl. Conf. on Mobile Data Management, MDM (2004)

15. Cohn, D., Dhoolia, P., Heath III, F., Pinel, F., Vergo, J.: Siena: From PowerPoint to Web App in 5 Minutes. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 722–723. Springer, Heidelberg (2008)
16. Boley, H., Paschke, A., Shafiq, O.: RuleML 1.0: The Overarching Specification of Web Rules. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) RuleML 2010. LNCS, vol. 6403, pp. 162–178. Springer, Heidelberg (2010)
17. Hollinsworth, D.: Workflow reference model. Technical report, Workflow Management Coalition, TC00-1003 (1994)
18. WebSphere Process Server, `http://www-01.ibm.com/software/ integration/wps/`
19. BizAgi Business Process Management System, `http://www.bizagi.com/`
20. ActiveVOS Business Process Management System, `http://www.activevos.com/`
21. Drools Expert, `http://www.jboss.org/drools/drools-expert.html`
22. Yongchareon, S., Liu, C., Zhao, X.: An Artifact-Centric View-Based Approach to Modeling Inter-Organizational Business Processes. In: Bouguettaya, A., Hauswirth, M., Liu, L. (eds.) WISE 2011. LNCS, vol. 6997, pp. 273–281. Springer, Heidelberg (2011)
23. Liu, R., Vaculín, R., Shan, Z., Nigam, A., Wu, F.: Business Artifact-Centric Modeling for Real-Time Performance Monitoring. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 265–280. Springer, Heidelberg (2011)

Appearance-Order-Based Schema Matching

Guohui Ding[1,3], Han Dong[2] and Guoren Wang[1,3]

[1] Key Laboratory of Medical Image Computing (NEU), Ministry of Education
[2] National Marine Data and Information Service
[3] College of Information Science & Engineering, Northeastern University, China
dgh_acheng@sina.com, donghan@mail.nmdis.gov.cn, wanggr@mail.neu.edu.cn

Abstract. Schema matching is widely used in many applications, such as data integration, ontology merging, data warehouse and dataspaces. In this paper, we propose a novel matching technique based on the order of attributes appearing in the schema structure of query results. The appearance order embodies the extent of the importance of an attribute for the user examining the query results. The core idea of our approach is to collect the statistics about the appearance order of attributes from the query logs to find correspondences between attributes in the schemas to be matched. As a first step, we employ a matrix to structure the statistics about the appearance order of attributes. Then, two scoring functions are considered to measure the similarity of the collected statistics. Finally, an traditional algorithm is employed to find the mapping with the highest score. Furthermore, our approach can be seen as a complementary member to the family of the existing matchers, and can also be combined with them to obtain more accurate results. We validate our approach with an experimental study, the results of which demonstrate that our approach is effective and has good performance.

1 Introduction

Schema matching plays an important role in the realm of data integration, which is a solution to sharing multiple heterogeneous data sources through a unified access interface. In essence, schema matching problem refers to the problem of finding semantic correspondences, also called *matches*, between elements of the source schema and elements of the target schema. The *match* means that its two elements hold the same meaning or refer to the same object. The *match* is very significant for creating a unified mediated schema over multiple source schemas, exchanging data from one schema to another schema and sharing data in the similar domain. The schemas to be matched are typically designed by different developers which have different habits and experiences, so they often have diverse structures and representations, and this makes schema matching difficult. Besides, dozens of tables and thousands of attributes in the schemas also increase the difficulty of schema matching. Even with some availability of domain expertise, the task of a schema matching may not be easy.

Much attention has been to paid to schema matching, and a multitude of techniques, also called *matchers*, have been proposed, e.g., [2,4,6,10,11]. However, these existing *matchers* are not infallible, because no *matcher* is perfect

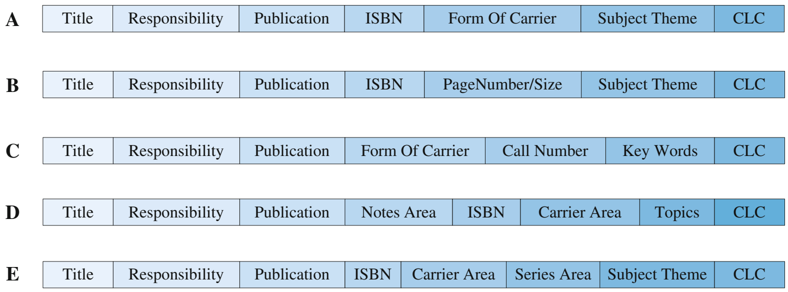
S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 79–94, 2012.
© Springer-Verlag Berlin Heidelberg 2012

and can return *matches* with 100% accuracy. Consequently, additional efforts are required for schema matching. In this paper, we proposed a novel matching technique that exploits the order of attributes appearing in the schema structure of query results to discover the *matches* between the attributes of the source schema and the attributes of the target schema. As is well known, the words in almost every book, we can read, are arranged from the left side to the right side, and this is a habit that people capture information from left to right. For example, given a spreadsheet of some books, people always start from the first column to read, then the second column, etc. As a result, the developers of the applications about the structured information always design the schema structure according to this habit. That is, the more important columns will be arranged at the positions closer to the left side. For example, the column "bookname" in the above spreadsheet may appear in the left-hand side of the column "author"; as such, the column "start time" will be arranged in the left side of the column "arriving time" in the railway timetable. The arrangement of these columns not only embodies the reading habit but also the default rule of some industry. We browse five digital libraries and pose the same query to their respective databases, then present the schema structures of their returned results about books in Figure 1. Surprisingly, all these libraries arrange the attributes of the book in almost the same order. It is easy to see that the attributes close to the left side are arranged according to the reading habit. However, the extent of the importance among the attributes close to the right side is almost the same, but they also have the similar order. The reason for this behavior is that these libraries fall into the same industry where there exist some default rules. Consequently, we are able to exploit the habits, which are typically conformed by both schemas to be matched, to find *matches*.

As is clear from the discussion above, different attributes have different importance of structuring the query results to be shown to the final users. As a result, an attribute will hold its own position in the schema structure of the query results. Actually, the appearance order of attributes refers to the positions of attributes appearing in the schema structure. Thus, the statistics about the appearance order of an attribute in a large number of query results can be seen as its identification differing from other attributes. Every query result corresponds to one query statement in the query log. The core idea of our approach is to collect the statistics about the appearance order of attributes from the query logs to find correspondences between attributes in the schemas to be matched. Our approach works in three steps. As the first phase, the query log of each schema is scanned to collect the statistics about the appearance order. We design two types of matrices to structure the statistics and call them feature matrices. One is used to record the information about the position of attributes, while the other is used to record the information about the number of attributes which are behind the current attribute. In the second phase, we consider three types of cardinality constraints for the mappings, which are one-to-one mapping, onto mapping and partial mapping. Then, two scoring functions are considered to measure the similarities of feature matrices of the schemas to be matched with respect to the three types of constraints. The task of the last phase is to employ a traditional searching method to find the attribute mapping with the highest score. Our approach can be seen as a complementary member to the family of existing *matchers* and can also be combined with them to achieve more accurate *match results*. This paper makes the following contributions:

1. We exploit the statistics about the appearance order of attributes in the schema structure of the query results to find *matches*.
2. Two types of feature matrices are employed to collect the statistics about the appearance order of the attributes from the query logs.
3. Two scoring functions are considered to measure the similarities of the feature matrices of the schemas to be matched.
4. We perform an extensive experimental study, the results of which show that the proposed algorithm has good performance.

The rest of this paper is organized as follows. Section 2 introduces the feature matrices. The scoring functions and the traditional searching algorithm are discussed in Section 3. The experimental results are given in Section 4. A brief related work is reviewed in Section 5. Finally, we conclude in Section 6.



A | Title | Responsibility | Publication | ISBN | Form Of Carrier | Subject Theme | CLC

B | Title | Responsibility | Publication | ISBN | PageNumber/Size | Subject Theme | CLC

C | Title | Responsibility | Publication | Form Of Carrier | Call Number | Key Words | CLC

D | Title | Responsibility | Publication | Notes Area | ISBN | Carrier Area | Topics | CLC

E | Title | Responsibility | Publication | ISBN | Carrier Area | Series Area | Subject Theme | CLC

Fig. 1. Schema Structure of Query Results from Five Digital Libraries A-E

2 Feature Matrices

In this section, we describe the main work of our first phase. Given two schemas to be matched, our main task is to scan the query log of each schema to collect the statistics about the appearance order of attributes. Then, two types of matrices are designed to structure the statistics collected from the query log.

As our motivation shows, the appearance order of an attribute in schema structure of query results can be seen as its identification differing from other attributes in the same schema. It is easy to think that just the clauses with the type “select” in the query log need to be considered, because the attributes in other types of clauses do not appear in the query results. However, just scanning the “select” clause itself is slightly incomplete. Consider the following example. For a developer who is designing the query interface of a website selling the mobile phones, it is natural to dispose the query condition “brand” ahead the condition “price”. If a user want to find the phones with brand “Nokia” and price under 3000 RMB, the website will produce a corresponding query with the “where” clause “brand=’Nokia’ and price <= ’3000”’, rather than “price <= ’3000’ and brand=’Nokia”’. If this scenario happens in another website also selling mobile phones, we may obtain the same “where” clause, because of people thinking in much the same way. We can see that the positions of attributes in the “where” clause can also identify themselves to some extent. As a result, we consider 4 types of clauses “select”, “where”, “group” and “order” during the process of scanning the query log. In addition to the types of the clauses, we need to consider the types of queries. As in [11], the following three types of queries are considered in our approach:

- SPJ: Single-block queries with Select, Project, Join and optional “group” and/or “order” clauses.
- SPJU: Multiple SPJ queries connected by the set operator “union”, but except “intersect”.
- SPJS: SPJ queries with nested subqueries falling into one of the three types.

For SPJ queries, it is a common process that involves creating the appearance sequences in Definition 1 below, then checking the position of each attribute in the appearance sequences, finally updating the corresponding entries in the matrix. For the SPJU and the SPJS queries, they are decomposed into separate subqueries each of which can be seen as a single-block SPJ query. Next, we will show the definition of the appearance sequence.

Definition 1. *Let* “select a_1,...,a_{n_1} from <table reference> [where b_1,...,b_{n_2}] [group by c_1,...,c_{n_3}] [order by d_1,...,d_{n_4}]” *be a query statement. Then, we call the sequence* $a_1...a_{n_1}[b_1...b_{n_2}][c_1...c_{n_3}][d_1...d_{n_4}]$ appearance sequence.

Based on the definition above, we can see that each query corresponds to an appearance sequence which embodies the reading habit of people and the default rules of some industry mentioned in Section 1. Now, the first task our approach is turned into collecting the statistics about the positions of attributes in the appearance sequence. Thus, each query in the query log is scanned to produce an appearance sequence. The position of each attribute in an appearance sequence is recorded in a matrix. The rows of the matrix represent all the attributes in one schema, while the columns represent the positions from 1 to the maximum of the number of elements in all the appearance sequences. An entry of the matrix represents the number of some attribute appearing in some position in all the appearance sequences. This is our first type of matrix, and we call it p-matrix. Before scanning the query log, all the entries in the p-matrix are initialized with the value 0. If an attribute appears in some position, the value of the corresponding entry in the p-matrix is incremented by one. After scanning all the queries in the query log, the entries in the p-matrix are normalized by dividing each of them by the largest entry. After this normalization step, the p-matrix is independent of the size of the query log. To understand the p-matrix intuitively, an example with dummy statistics and six dummy attributes a - e is shown in the left-hand panel of Figure 2, while the normalized one is shown in the other side.

Consider two appearance sequences $S_1{=}a_1a_2a_3a_4a_5$ and $S_2{=}b_1b_2$. Although the two attributes a_1 and b_1 are both in the first position, we believe that a_1 is greater than b_1 in terms of the importance of structuring the query results to be shown to the final users. This is because a_1 ranks prior to four attributes in its sequence, nevertheless just one attribute behind b_1. We can see that just the statistics about the position is not enough. As a result, we make a little change to the original matrix to collect the information about the number of attributes which rank after a given attribute in the appearance sequence. The changed matrix is called n-matrix, which is our second type of matrix. Actually, the n-matrix is similar to the p-matrix, and they have the same columns and same rows. The difference between them is that each time increasing the value of the corresponding entry for an attribute e, the increment is not the value 1 but rather the number of the attributes which rank after e in the appearance sequence. Except the information about the positions of attributes, the n-matrix captures a little more information than the p-matrix, and their performance is tested and compared in our experiment. Now, given tow schemas to be matched, we can obtain two corresponding matrices. The two matrices can be seen as the respective feature of attributes in the two schemas. Thus, our task of matching attributes is transformed into measuring the similarity of the two matrices. In the next section, two scoring functions are introduced for the measurement of the similarity of the two matrices.

$$
\begin{array}{c|c|c|c|c|c|}
 & 1 & 2 & 3 & 4 & 5 \\ \hline
a & 952 & 0 & 0 & 211 & 91 \\ \hline
b & 24 & 899 & 0 & 188 & 52 \\ \hline
c & 0 & 0 & 907 & 163 & 0 \\ \hline
d & 0 & 0 & 0 & 877 & 0 \\ \hline
e & 533 & 0 & 0 & 0 & 911 \\ \hline
\end{array}
\xrightarrow{\text{Normalized (/952)}}
\begin{array}{c|c|c|c|c|c|}
 & 1 & 2 & 3 & 4 & 5 \\ \hline
a & 1.000 & 0.000 & 0.000 & 0.221 & 0.096 \\ \hline
b & 0.025 & 0.944 & 0.000 & 0.197 & 0.055 \\ \hline
c & 0.000 & 0.000 & 0.953 & 0.171 & 0.000 \\ \hline
d & 0.000 & 0.000 & 0.000 & 0.921 & 0.000 \\ \hline
e & 0.560 & 0.000 & 0.000 & 0.000 & 0.957 \\ \hline
\end{array}
$$

Fig. 2. An Example of p-matrix

3 Scoring Functions and Search Algorithm

In this section, our discussion is divided into two parts. Two scoring functions are discussed in the first subsection, while the search algorithm is discussed in the second subsection.

3.1 Scoring Functions

Before the main discussion, we introduce several types of cardinality constraints in schema matching, which are the prerequisite of the scoring functions. In our approach, we consider three types of cardinality constraints: one-to-one mapping, onto mapping and partial mapping, which are first proposed in [4]. For two input schemas S_1 and S_2 to be matched, the three types of cardinality constraints are described as follows:

- One-to-one mapping: For each attribute in S_1, there exists one and only one corresponding attribute as the counterpart in S_2, and vice versa. If the two schemas S_1 and S_2 are referred to as two sets and the mapping is referred to as a function, we can see that this function is the so-called bijective mapping in discrete mathematics, i.e., it is both surjection and incidence.
- Onto mapping: For each attribute in S_1, there exists unique attribute in S_2 as a *match* . Conversely, each attribute in S_2 either has one and only one attribute in S_1 as a *match* or remain unmatched. Compared to the one-to-one mapping, this mapping actually falls into the class of the incidence.
- Partial mapping: Each attribute in S_1 either has one and only one attribute in S_2 as a *match* or remain unmatched, and vice versa. In practice, this case is the most general and difficult one. The reason is that for an attribute in one schema, the existence of its *match* (counterpart) in another schema is unknown (uncertain); for a schema, the number of its attributes which have *matches* in another schema is unknown.

This three types of cardinality constraints are very prevalent in practice, as opposed to the case where an attribute in one schema has multiple *matches* in another schema, so we do not consider this class of cardinality constraints.

The problem of how to create an effective scoring function to evaluate the quality of matching has been discussed in [4]. They proposed two scoring functions and addressed the problem of the monotonicity of the scoring functions. They classified their scoring functions into *monotonic* and *non-monotonic*. Given the mapping, we exploit their scoring functions as the measurement of similarity of feature matrices in our approach. As in [11], we introduce some formal descriptions about schema matching. Let S_1 and S_2 be two schemas to be matched. Given their respective feature matrices, the matching task is to find a optimal mapping $\hat{m}$ that gives the highest score for a specific scoring function. Consequently, any mapping m should provide three kinds of information for the scoring function. The first is the number of the *matches* (matching attributes) included in m, denoted as k_m; the second is the attributes occurring in m, denoted as $\{a_1, ..., a_i, ..., a_{k_m}\}$ for S_1 and $\{b_1, ..., b_j, ..., b_{k_m}\}$ for S_2; and the third is the actual correspondences between attributes of S_1 and S_2, i.e., $m(a_i) = b_j$ $(m(i) = j)$. Here, it should be noted that k_m is the number of the *matches* in m rather than the number of the correct *matches* in m. For two feature matrices, they are required to have the same number of the rows and columns for the computation of the similarity. Thus, if the number of their rows and columns is not equal, additional rows and columns with values 0 are added into the end of the corresponding matrix. Now, we present the definition of the *monotonic* scoring function.

Definition 2. *Let P_1 and P_2 be the two feature matrices collected from the query logs of schema S_1 and S_2 respectively, and n be the number of the rows of P_1 and P_2. Let a_{ij} be an entry in P_1, which represents the statistics about attribute a_i appearing in position j, while b_{ij} be an entry in P_2, which represents the statistics about attribute b_i appearing in position j. Given a mapping m, the monotonic scoring function is defined as:*

$$f_e(m) = 1 - \frac{1}{ub}\sqrt{\sum\nolimits_{i=1}^{k_m}\sum\nolimits_{j=1}^{n}(a_{ij} - b_{m(i)j})^2} \tag{1}$$

Given the mapping m, this scoring function employs the Euclidean distance metric to measure the similarity between two feature matrices. Let $d_m(P_1, P_2)$ be the Euclidean distance above, i.e., the square root item. We can see that $d_m(P_1, P_2)$ increases monotonically with the increase of the number of *matches* in m; that is, this function is *monotonic* in k_m. Given two schemas to be matched, if the correct k_m is unknown, the matching algorithm using this function will just return the mapping with only one *match* as $\hat{m}$, because the score of any mapping with more than one *match* will be smaller than the one with only one *match*. As a result, this function can be used to achieve the one-to-one mapping and the onto mapping problems, rather than the partial mapping problem. The variable ub takes the value $\sqrt{k_m * n}$ that is the upper bound to the value of $d_m(P_1, P_2)$; and this guarantees that the value of the function is positive. In the following, we will discuss the *non-monotonic* scoring function.

Definition 3. *Let P_1 and P_2 be the two feature matrices with n rows. Let a_{ij} be an entry in P_1, which represents the statistics about attribute a_i appearing in position j, while b_{ij} be an entry in P_2, which represents the statistics about attribute b_i appearing in position j. Given a mapping m, the non-monotonic scoring function is defined as:*

$$f_n(m) = \sum\nolimits_{i=1}^{k_m}\sum\nolimits_{j=1}^{n}(1 - \alpha\frac{|a_{ij} - b_{m(i)j}|}{a_{ij} + b_{m(i)j}}) \tag{2}$$

Now, we will analyze the principle of this scoring function. The item multiplied by α in the equation above is the *normal distance* [4]. Suppose that if the statistics values about the position of attributes are uniformly distributed and two of them are randomly chosen from the matrices, the expected value of *normal distance* is β (around $\frac{1}{3}$). As a result, if the control parameter α is set to $\frac{1}{\beta}$ (3), the expected score of this function becomes 0 with the assumption above. In other words, the *match* of randomly chosen two attributes will not contribute to the final score in such cases. In contrast, if the *match* is correct (the two attributes map correctly), it will positively contribute to the final score. It can be seen that for a mapping m, the more correct *matches* m includes, the higher score m is rewarded. Thus, the optimal mapping $\hat{m}$ is expected to be rewarded the highest score among other mappings. Based on the analysis above, we can see that this function is non-monotonic in k_m. Actually, the value $\frac{1}{\alpha}$ represents the average of the *normal distance* or the approximate demarcation point between the *normal distance* of the correct *matches* and the one of the wrong *matches*. The control α can be computed via the quantile or the experiments. Further, the behavior of the scoring function can be controlled by changing the parameter α (see [4] for more details).

3.2 Search Algorithm

Given the feature matrices and the scoring functions, now, our task is to find the optimal attribute mapping. In this section, we first introduce how to refer to the problem of searching the optimal mapping as the combinatorial optimization problem, then present the details of the search algorithm.

Given two schemas S_1 and S_2 to be matched, the first schema S_1 has n_1 attributes $\{a_1, ..., a_i, ..., a_{n_1}\}$, while S_2 has n_2 attributes $\{b_1, ..., b_j, ..., b_{n_2}\}$. Consider the first cardinality constraint one-to-one mapping where $n_1 = n_2$. If the attributes of S_1 are regarded as a fixed sequence $a_1a_2...a_{n_1}$ and the correspondence is fixed $m(a_i) = b_i$, any instance of the permutation of all attributes of S_2 corresponds to a possible mapping. For example, if $n_1 = n_2 = 2$, then the permutation b_1b_2 and b_2b_1 correspond to two possible mappings $\{(a_1, b_1), (a_2, b_2)\}$ and $\{(a_1, b_2), (a_2, b_1)\}$, where each mapping includes two *matches*. As a result, we can see that our task of finding the $\hat{m}$ can be transformed into the combinatorial optimization problem where the score of each permutation is the score of its corresponding mapping. However, for another two cardinality constraints onto-mapping and partial mapping, the numbers of the attributes of the two schemas are typically not equal. To perform the problem transformation above, we need to make the two schemas own the same number of attributes n. For this purpose, the "dummy" attributes [11] are added to S_1 and S_2. The problem of how many attributes should be added depends on the scoring function. If the monotonic function is used, the $n_2 - \hat{k}_m$ attributes will be added to S_1, while $n_1 - \hat{k}_m$ attributes will be added to S_2, so each schema has $n_1 + n_2 - \hat{k}_m = n$ attributes. Here, for onto mapping $\hat{k}_m$ is also known and takes the value $\min(n_1, n_2)$, but for partial mapping it is the estimate of the number of the correct *matches* between S_1 and S_2 and should be given to the algorithm. Now, we describe how to decide the number of attributes added. For S_2, there exist $n_2 - \hat{k}_m$ attributes $\{b_q...b_r\}$ which have no matching attributes in S_1, so $n_2 - \hat{k}_m$ "dummy" attributes are added to S_1 as the matching attributes for $\{b_q...b_r\}$. The reason for S_1 is the same as S_2, thus we gives unnecessary details no longer. For the non-monotonic function, the $\hat{k}_m$ is not required, so it is considered to be zero. Thus, n_2 "dummy" attributes will be added to S_1, while n_1 "dummy" attributes will be added to S_2. We can see that in addition to making the two schemas have the same number of attributes, another purpose of the "dummy" attributes is to make each attribute have a counterpart in the other schema. Here, it should be noted that the *matches* that involve the "dummy" attributes are ignored, when the search algorithm computes the score of a mapping.

We can see that while regarding the attributes of S_1 as a fixed sequence, the number of the permutations of all attributes of S_2 is $n!$. The space of all the

Algorithm 1. Generator of New Solutions

```
input  : $p = b_1 b_2 ... b_i ... b_n$, a permutation, the current solution;
output: $p'$, the new solution;
int $q, r = random()$; //randomly generate two numbers, $q, r < n$
if $q < r$ then
    $p' = b_1 b_2 ... b_{q-1} b_r b_{r-1} ... b_{q+1} b_q b_{r+1} ... b_n$;
if $q > r$ then
    $p' = b_r b_{r-1} ... b_1 b_{r+1} ... b_{q-1} b_n b_{n-1} ... b_{q+1} b_q$;
if $q == r$ then
    $p' = b_q b_{q-1} ... b_1 b_{q+1} ... b_n$;
return $p'$
```

permutations is very large and an exhaustive search is not feasible. Thus, we exploit the simulated annealing (SA) algorithm [1], which is the classical solution to the combinatorial optimization problem, to find the optimal permutation corresponding to $\hat{m}$, denoted by $\hat{p}$. SA algorithm is the random search technique based on physical annealing process, which can approach the global optimization gradually by continuously breaking off from the local optimization. In our context, the scoring functions are used as the energy function of SA. It should be noted that we aim at the solution with the highest score rather than the usual lowest energy. The SA algorithm involves six key components: the generator of the new solutions, the Metropolis criterion, the initial temperature, the length of Markov chain, the temperature-fall period and the stopping criteria. We now briefly explain each of these components in our context.

The implementation of the generator is shown in Algorithm 1. Given a current solution $b_1 b_2 ... b_n$, two numbers q and r are randomly generated. If $q < r$, the elements from b_q to b_r are re-arranged in reverse order, while the order of other elements remain unchanged. If $q > r$, the elements from b_1 to b_r and from b_q to b_n are conversely re-arranged respectively. If $q = r$, the elements from b_1 to b_q are conversely re-arranged. Here, other methods can also be used, for example the *cross* and *mutation* leveraged from the genetic algorithm.

The Metropolis criterion represents the acceptance criterion of the new solutions. Let p_1 be the current solution, p_2 be the new solution and $f(x)$ be the scoring function, i.e., $f_e(x)$ or $f_n(x)$. The Metropolis criterion means that if $f(p_2) > f(p_1)$, using p_2 as the current solution instead of p_1, else if $\exp(-\frac{f(p_1)-f(p_2)}{T}) > rn$, also accepting p_2, else preserving p_1 and abandoning p_2, where rn is a random number in range $(0, 1)$ and T is the current temperature. SA algorithm makes use of this criterion to make a decision about whether or not to accept the new solutions.

The initial temperature T_0 of SA is central to obtainment the global optimization, and the higher the T_0 is, the closer the solutions approach the real solution. However, the much higher T_0 will lead to the unacceptable running time. The work [1] proposed that the T_0 should enable the acceptance rate of the new solutions to approach the value 1 at the beginning. This means that the acceptance possibility of Metropolis criterion is close to 1 at the beginning,

Algorithm 2. Search Algorithm

```
   input : $n$, the number of the attributes in the solution;
           $p$, a permutation, a solution;
           $T$, the temperature;
   output: $\hat{p}$, the optimal solution;
 1 initialize $p$, $T$; // $p$ is initialized to a random solution
 2 repeat
 3     int $length = 0$;
 4     repeat
 5         $p' = generator(p)$; // generate the new solution
 6         if $f(p') > f(p)$ then
 7             $p = p'$;
 8         else
 9             $rn$ = random($< 1$); // randomly generate a number in range $(0, 1)$
10             if $\exp(\frac{f(p')-f(p)}{T}) > rn$ then
11                 $p = p'$;
12         $length$++;
13     until $length > 10n$
14     $T = 0.95T$;
15 until stopping criterion
16 return $\hat{p} = p$;
```

i.e., $\exp(\frac{-\triangle f}{T_0}) \approx 1$. In practice, this possibility is topically set to 0.95. To use this method to compute the T_0 in our context, we need to compute the $\triangle f$. Here, the statistical method is used to compute the score difference. We randomly choose k pairs of solutions ($k > 1000$), then compute the score difference of each pair, finally, take the expectation of these differences as the value of $\triangle f$.

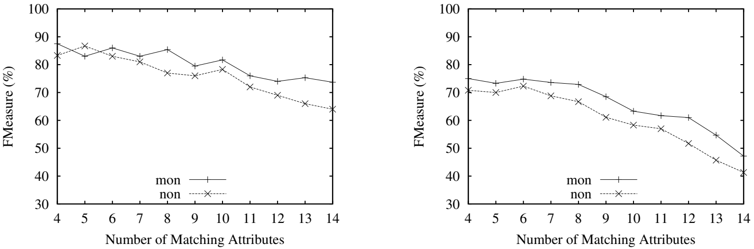
The last three components of SA are simple relatively. The length of the Markov chain is topically associated with the problem size, and the too long chains would not help find the global optimization [1]. Thus, the length of Markov chain is set to $10n$ in our approach. For the temperature-fall period, we make use of the classical method, i.e., $T_{k+1} = \beta T_k$, to control the attenuation of the temperature, where $\beta = 0.95$ and T_k is the current temperature. During the running of the algorithm, if the consecutive r Markov chains have no improvement on the current optimization, the search process will terminate; this is the stopping criterion. Now, based on these components, the details of the search algorithm are shown in Algorithm 2.

The algorithm begins with a random solution (line 1), because the initial solution has very little effect on its performance. Then, it randomly explores the new solutions (line 5), and employs the Metropolis criterion to decide whether to update the current solution (lines 6-11). Actually, the internal circle corresponds to a Markov chain (lines 4-13). Thus, each iteration of the external circle will generate a Markov chain. If the length exceeds $10n$, the spread of the chain is over (line 13), and the temperature falls for the next chain (line 14). If after r iterations of the external circle the current solution remains unchanged, the algorithm will terminate and return the current solution as the $\hat{p}$.

4 Experimental Evaluation

In this section, we test the time cost and evaluate the quality of the matching results of our proposed approach in synthetic schema matching scenarios. First, we present how to generate the synthetic data set used in the experiments. Then, we show the experimental results evaluating the performance of our matching algorithm in the case of the three cardinality constraints (one-to-one, onto and partial). We also study the effect of varying the control parameter of the non-monotonic function on the performance of our approach. Finally, we test the time cost of the proposed algorithm. Our algorithm is implemented using C++ language and the experiments are carried on a PC compatible machine, with Intel Core Duo processor (2.33GHz).

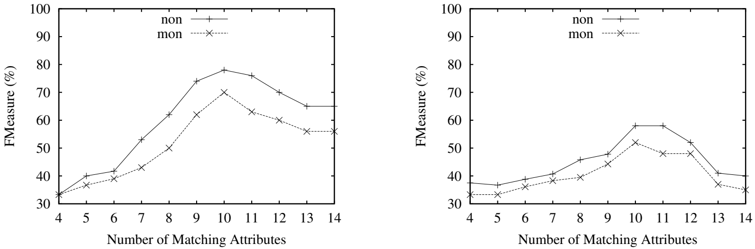
We produce the experimental data set based on two online bookstores developed by different persons. The schema of the first bookstore includes 31 attributes, while the second includes 35 attributes, and there exist 27 matching attributes (*matches*) of each schema. We suppose that a fictitious user continuously accesses the bookstore until the produced log including 8000 queries. These SQL statements include two kinds of queries: the random queries with any keywords according to the query interface of the bookstore and the fixed queries generated based on the navigation or the classification functions of the bookstore. Based on the query logs, two feature matrices of the two bookstores can be obtained as the experimental data. In the experiments, the FMeasure metrics is used as the measurement of the performance of the algorithm.



(a) n-matrix (b) p-matrix

Fig. 3. Results of One-to-one Mappings

We evaluate the accuracy against the correct mappings determined by manual inspection of the source and target schemas. We run our algorithm with randomly chosen subset of the experimental data in each experiment for many times, then get the average of the experimental results. We first present the results of one-to-one mapping in Figure 3. We use the notation "mon" and "non" as the shorthand for the monotonic function and the non-monotonic function respectively. The control parameter α for non-monotonic function is set to 0.3. As it can be seen in Figure 3(a), the *match* results gradually deteriorate as the number of the matching attributes (*matches*) increases, and the worst results for "non" function is nearly 60%. The results with "mon" are better than the one with "non". This is because the correct k_m is known for the one-to-one mapping. We can also see

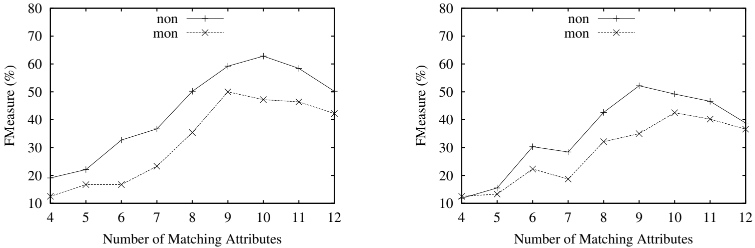


(a) n-matrix (b) p-matrix

Fig. 4. Results of Onto Mappings

that the overall quality of the results over the n-matrix are higher than the one over p-matrix. The reason is that the information collected in the n-matrix is more than the information in the p-matrix.

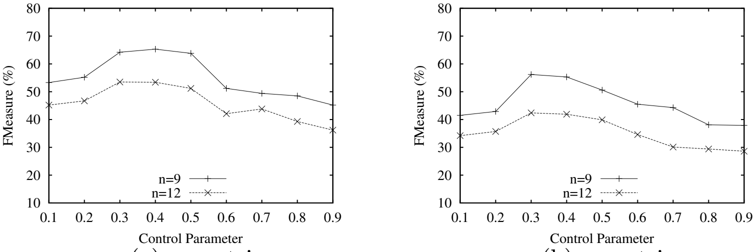
The experimental results corresponding to the onto cardinality constraint are shown in Figure 4. Here, the size of the target schema is kept constant at 16 attributes while the matching attribute number of the source schema is increased from 4 to 14. As it can be seen, in both data sets, the results with “non” outperform the results with “mon”. The accuracy with “non” reaches 80% in Figure 4(a) while it was 59% in Figure 4(b). And the overall quality over the n-matrix is better than the p-matrix. However, the accuracy in the onto mapping case gradually improves as the number of the matching attributes increases, then the accuracy begins to decline when the number exceeds some value; this is just contrary to the one-to-one mapping. The reason is that the matching with the onto constraint requires extra effort in contrast to the one-to-one case, i.e., the onto mapping first need to choose a subset of the 16 attributes, then the onto mapping based on the chosen attribute subset is turned into the one-to-one mapping.



(a) n-matrix (b) p-matrix

Fig. 5. Results of Partial Mappings

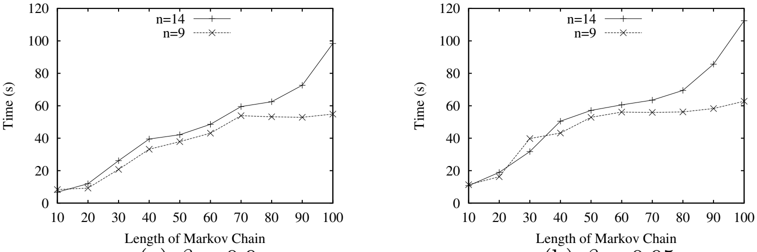
Figure 5 illustrates the results of the matching with the partial mapping cardinality constraint. In this experiment, we fix the size of both source and target schema at 14, and vary the number of the matching attributes from 4 to 12. To enable the experiment with the “mon” in the partial mapping case, we give the number of the correct *matches* to the algorithm with “mon” function. Here, the trend of curves in Figure 5 is similar to the above experiment, but the best performance is less than 70%. It can be seen that the matching with the partial cardinality constraint is the most difficult matching. Conversely, the results with "non" function are better than the one with the "mon" function.



(a) n-matrix (b) p-matrix

Fig. 6. Varying the Control Parameter α

Now, we test the effect of varying the control parameter α on the *match* results. We fix the size of both source and target schema at 14 attributes, and fix the number of the correct *matches* at 9 and 12 respectively, denoted by $n = 12$ and $n = 9$. The experiment results are shown in Figure 6. We can see that the accuracy first increases as α increases from 0.1 to 0.3, then achieves the highest value as $\alpha \in (0.3, 0.5)$, finally drops with the increase of α. The accuracy with $n = 9$ is higher than the one with $n = 12$, which is consistent with the above experiments.



(a) $\beta = 0.9$ (b) $\beta = 0.95$

Fig. 7. Time Cost

Finally, we test the time cost of our algorithm with one-to-one cardinality constraint. In this experiment, we set the number of the correct *matches* to 9 and 14 respectively, denoted by $n = 9$ and $n = 14$, and set the temperature-fall coefficient $\beta = 0.9$ and $\beta = 0.95$. The results are shown in Figure 7. The time with $n = 14$ increases as the length of the Markov chain increases, and the biggest running time reaches nearly two minutes. However, the time cost with $n = 9$ remains unchanged after the length beyond 60. The reason for this behavior is that the size of the search space for $n = 9$ is less than the number of all iterations of the algorithm, so the algorithm will accomplish the search process ahead of time and ignore the following iterations caused by the increase of the length. It can also be seen that the time cost for $\beta = 0.9$ in Figure 7(a) is less than the cost for $\beta = 0.95$ in Figure 7(b). The reason is that the number of iterations increases for $\beta = 0.95$.

5 Related Work

Schema matching has been an active research field for a long time [2,4,6,8,11]. A survey of approaches to automatic schema matching is presented in work [2]. They present a taxonomy that covers many of these existing approaches, and describe the approaches in some detail. These existing techniques are called *matchers* by their work and are mainly classified as schema-based and instance-based. Schema-based *matchers* only consider schema information that includes the usual properties of schema elements, such as name, description, data type, relationship types, constraints, and schema structure. Instance-based *matchers* can give important insight into the data stored in the schemas, especially in the case that useful schema information is limited.

The work [4] proposes an approach which is fitting into the situation that the column names in the schemas and the data in the columns are "opaque" or very difficult to interpret. Their technique works in two steps. First, they measure the pair-wise attribute correlations in the tables to be matched via using the *mutual information*. Then, they find matching node pairs between the dependency graphs by a heuristic algorithm. A recent work [8] puts the context into schema matching in order to improve the quality of data exchange. The context actually refers to the categorical attribute whose values are discrete. These attributes can classify the source instances into different categories. They make use of the categorical attributes as the constraint to restrict the *matches* to work only for partial data instances in the same relation.

A versatile graph matching algorithm named "similarity flooding" is proposed in work [3]. The key idea of their method is the assumption that whenever any two nodes in the graphs are found to be similar, the similarities of their adjacent nodes increase. Thus, the similarity between two nodes is computed as the sum of their own similarities plus their neighbors'. After some iterations, the initial similarity of any two nodes propagates through the graph, i.e., similarity flooding. Corpus-based Schema Matching is proposed in work [6]. They show how a corpus of schemas and mappings can be used as a new resource for identifying the attributes in schema matching. They exploit such a corpus in two ways. The first is to learn the variation and the similar properties of the element to be matched from the corpus, while the second is to learn the statistics about elements and their relationships and use them to infer constraints.

A new class of techniques, called usage-based schema matching, is proposed in the recent work [11]. Their key idea is to exploit the feature extracted from the query log to find the correspondence of the attributes. They identify co-occurrence patterns which represent that two attributes appear in the two query clauses together. Finally, they employ the genetic algorithm to find the highest-score mappings. Recently, different from the traditional techniques, the possible mapping is introduced to schema matching [10], which presents another research way for schema matching. For an attribute, the possible mapping represents that there are multiple matching candidates with respect to this attribute. They use the possible mappings to create the possible mediated schemas to retrieve multiple possible query results for one query.

6 Conclusion

In this paper, we employ the order of attributes appearing in the schema structure of query results to perform schema matching. The appearance order embodies the extent of the importance of an attribute for the user examining the query results. We first collect the statistics about the appearance order of attributes from the query logs of the schemas to be matched. Then, two types of matrices are designed to structure the statistics about the appearance order of attributes. The first one is called p-matrix while the second one is called n-matrix. Third, two scoring functions are considered to measure the similarity of the collected statistics. One function is monotonic with the number of the correct *matches* and fit the one-to-one and onto mappings, while the other is non-monotonic and is designed for partial mappings. Finally, the simulated annealing (SA) algorithm is employed to find the mapping with the highest score. We perform extensive experiments to test the proposed approach, and the experimental results show that our approach performs well.

Acknowledgments. This research was supported by the State Key Program of National Natural Science of China (Grant No. 60933001), the National Science Foundation for Distinguished Young Scholars of China (Grant No. 61025007) and the National Natural Science Foundation of China (Grant No. 61100024 60803026 and 60973020).

References

1. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science Journal 220, 671–680 (1983)
2. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal 10(4), 334–350 (2001)
3. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: a versatile graph matching algorithm. In: Proc. of ICDE, pp. 117–128 (2002)
4. Kang, J., Naughton, J.F.: On Schema Matching with Opaque Column Names and Data Values. In: Proc. of SIGMOD, pp. 205–216 (2003)
5. Franklin, M., Halevy, A., Maier, D.: From Databases to Dataspaces: A New Abstraction for Information Management. In: Proc. of SIGMOD, pp. 1–7 (2005)
6. Madhavan, J., Bernstein, P., Doan, A., Halevy, A.: Corpus based schema matching. In: Proc. of ICDE, pp. 57–68 (2005)

7. Halevy, A., Rajaraman, A., Ordille, J.: Data integration: The teenage years. In: Proc. of VLDB, pp. 9–16 (2006)
8. Bohannon, P., Elnahrawy, E., Fan, W., Flaster, M.: Putting context into schema matching. In: Proc. of VLDB, pp. 307–318 (2006)
9. Warren, R.H., Tompa, F.: Multicolumn Substring Matching for Database Schema Translation. In: Proc. of VLDB, pp. 331–342 (2006)
10. Dong, X., Halevy, A.Y., Yu, C.: Data integration with uncertainty. In: Proc. of VLDB, pp. 687–698 (2007)
11. Chan, C., Elmeleegy, H.V.J.H., Ouzzani, M., Elmagarmid, A.: Usage-Based Schema Matching. In: Proc.of ICDE, pp. 20–29 (2008)

Fast Result Enumeration for Keyword Queries on XML Data

Junfeng Zhou[1], Zhifeng Bao[2], Ziyang Chen[1], and Tok Wang Ling[2]

[1] School of Information Science and Engineering, Yanshan University
[2] School of Computing, National University of Singapore
{zhoujf,zychen}@ysu.edu.cn, {baozhife,lingtw}@comp.nus.edu.sg

Abstract. In this paper, we focus on efficient construction of *tightest matched subtree (TMSubtree)* results for keyword queries on XML data based on SLCA semantics, where "matched" means that all nodes in a returned subtree satisfy the constraint that the set of distinct keywords of the subtree rooted at each node is not subsumed by that of any of its sibling node, while "tightest" means that no two subtrees rooted at two sibling nodes can contain the same set of keywords. Assume that d is the depth of a given TMSubtree, m is the number of keywords of a given query Q, we proved that if $d \leq m$, a matched subtree result has at most $2m!$ nodes; otherwise, the size of a matched subtree result is bounded by $(d-m+2)m!$. Based on this theoretical result, we propose a pipelined algorithm to construct TMSubtree results *without* rescanning all node labels. Experiments verify the benefits of our algorithm in aiding keyword search over XML data.

1 Introduction

Over the past few years, keyword search on XML data has been a hot research issue along with the ever increase of XML-based applications [3, 5, 7, 8, 9, 11, 13, 14, 15, 16, 17]. Same as the importance of effectiveness to keyword search, efficiency is also a key factor to the success of keyword search.

Typically, an XML document is modeled as a node-labeled tree T. For a given keyword query Q, each result t is a subtree of T containing each keyword of Q at least once, where the root node of t should satisfy a certain semantics, such as SLCA [15], ELCA [5, 16, 17], VLCA [8] or MLCA [9]. Based on the set of qualified root nodes, there are three kinds of subtree results: (1) *complete subtree (CSubtree)*, which is a subtree t_v^C rooted at a node v that is excerpted from the original XML tree without pruning any information [5, 15]; (2) *path subtree (PSubtree)*, which is a subtree t_v^P that consists of paths from v to all its descendants, each of which contains at least one input keyword [6]; (3) *matched subtree (MSubtree)*, which is a subtree t_v^M rooted at v satisfying the constraints of *monotonicity* and *consistency* [7, 11]. Let $S_{v'} = \{k_1, k_2, ..., k_{m'}\}$ be the set of distinct keywords contained in the subtree rooted at a node v', $S_{v'} \subseteq Q$. Intuitively, a subtree t_v is an MSubtree if and only if for any descendant node v' of v, there does not exist a sibling node u' of v', such that $S_{v'} \subset S_{u'}$, which

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 95–109, 2012.
© Springer-Verlag Berlin Heidelberg 2012

we call as the constraint of "*keywords subsumption*". Obviously, for a given CSubtree t_v^C, t_v^P can be got by removing from t_v^C all nodes that do not contain any keyword of the given query in their subtrees, and according to [11], t_v^M can be got by removing from t_v^P all nodes that do not satisfy the constraint of keywords subsumption.

Example 1. To find for "CS" laboratory all publications that are written by "Tom" and published in "DASFAA" about "XML" from the XML document D in Fig. 1, we may submit a query $Q = \{\text{CS, Tom, DASFAA, XML}\}$ to complete this task. Obviously, the qualified SLCA node is the root node with Dewey [14] label "1". Therefore, the *CSubtree* for Q is D itself, the *PSubtree* is R_1, while the *MSubtree* is R_2.

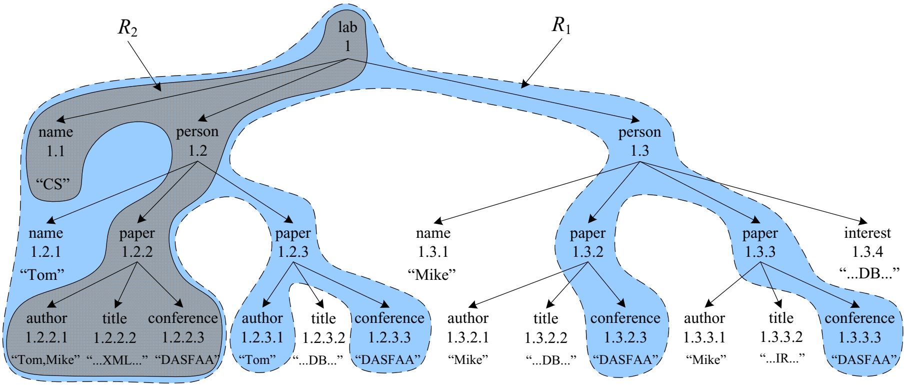


Fig. 1. A sample XML document D

From Example 1 we know that a *CSubtree*, e.g. D, may be incomprehensible for users since it could be as large as the document itself, while a *PSubtree* could make users feel frustrated since it may contain too much irrelevant information, e.g., although each leaf node of R_1 directly contains at least one keyword of Q, the three papers with Dewey labels "1.2.3, 1.3.2, 1.3.3" have nothing to do with "XML". In fact, from Fig. 1 we can easily know that for "CS" laboratory, the paper written by "Tom" and published in "DASFAA" about "XML" is the node with Dewey label "1.2.2". According to Fig. 1, we know that the keyword sets for node 1.1, 1.2 and 1.3 are $S_{1.1} = \{\text{CS}\}$, $S_{1.2} = \{\text{Tom, DASFAA, XML}\}$, and $S_{1.3} = \{\text{DASFAA}\}$, respectively. According to the constraint of keywords subsumption, all nodes in the subtree rooted at node 1.3 should be pruned, since $S_{1.3} \subset S_{1.2}$. Similarly, the keyword sets for node 1.2.1, 1.2.2 and 1.2.3 are $S_{1.2.1} = \{\text{Tom}\}$, $S_{1.2.2} = \{\text{Tom, DASFAA, XML}\}$ and $S_{1.2.3} = \{\text{Tom, DASFAA}\}$, respectively. According to the constraint of keywords subsumption, since $S_{1.2.1} \subset S_{1.2.2}$ and $S_{1.2.3} \subset S_{1.2.2}$, all nodes in the subtrees rooted at node 1.2.1 and 1.2.3 should be removed. After that, we get the *MSubtree*, i.e., R_2, which contains all necessary information after removing nodes that do not satisfy the constraint of keywords subsumption, and is more self-explanatory and compact.

However, most existing methods [2, 13, 15, 16, 17] addressing efficiency focus on computing qualified root nodes, such as SLCA or ELCA nodes, as efficient as possible. In fact, constructing subtree results is not a trivial task. Existing methods [7, 11] need to firstly *scan* all node labels to compute qualified SLCA/ELCA results, then *rescan* all node labels to construct the initial subtrees. After that, they need to buffer these subtrees in memory and apply the constraint of keywords subsumption on each node of these subtrees to prune nodes with keyword sets subsumed by that of their sibling nodes, which is inefficient in time and space.

As illustrated by [7], an MSubtree could still contain redundant information. E.g., the four conference nodes, i.e., 1.2.2.3, 1.2.3.3, 1.3.2.3 and 1.3.3.3, of D in Fig. 1 are same to each other according to their content, and for keyword query {CS, conference}, returning only one of them is enough. However, the MSubtree result contains all these conference nodes, because all of them satisfy the constraint of keywords subsumption.

In this paper, we focus on constructing *tightest matched subtree (TMSubtree)* results according to SLCA semantics. Intuitively, a TMSubtree is an MSubtree after removing redundant information, and it can be generated from the corresponding PSubtree by removing all nodes that do not satisfy the constraint of keywords subsumption and just keeping one node for a set of sibling nodes that have the same keyword set. Assume that d is the depth of a given TMSubtree, m is the number of keywords of the given query Q, we proved that if $d \leq m$, then a TMSubtree has at most $2m!$ nodes; otherwise, the number of nodes of a TMSubtree is bounded by $(d - m + 2)m!$. Based on this theoretical result, we propose a pipelined algorithm to compute TMSubtrees *without* rescanning all node labels. Our algorithm sequentially processes all node labels in document order and immediately outputs each TMSubtree once it is found. Compared with the MaxMatch algorithm [11], our method reduces the space complexity from $O(d \sum_1^m |L_i|)$ to $O(d \cdot max\{2m!, (d-m+2)m!\})$, where L_i is the inverted Dewey label list of keyword k_i.

The rest of the paper is organized as follows. In Section 2, we introduce background knowledge and discuss related work. In Section 3, we give an in-depth analysis to the MaxMatch algorithm[11], then define the tightest matched subtree (TMSubtree) and discuss its properties, and finally, present our algorithm on computing all TMSubtree results. In Section 4, we present the experimental results, and conclude our paper in Section 5.

2 Background and Related Work

We model an XML document as a node labeled ordered tree, where nodes represent elements or attributes, while edges represent direct nesting relationship between nodes in the tree. Fig. 1 is a sample XML document. We say a node v directly contains a keyword k, if k appears in the node name or attribute name, or k appears in the text value of v.

A Dewey label of node v is a concatenation of its parent's label and its local order, the last component is the local order of v among its siblings whereas the sequence of components before the last one is called parent label. In Fig. 1, the Dewey label of each node is marked as the black sequence of components separated by ".". For a Dewey label $A : a_1, a_2, ..., a_n$, we denote the number of components of A as $|A|$, and the i^{th} component as $A[i]$. As each Dewey label [14] consists of a sequence of components representing the path from the document root to the node it represents, Dewey labeling scheme is a natural choice of the state-of-the-art algorithms [1, 5, 10, 13, 15] for keyword query processing on XML data. The positional relationships between two nodes include Document Order ($\prec_d$), Equivalence ($=$), AD (ancestor-descendant, $\prec_a$), PC (parent-child, $\prec_p$), Ancestor-or-self ($\preceq_a$) and Sibling relationship. $u \prec_d v$ means that u is located before v in document order, $u \prec_a v$ means that u is an ancestor node of v, $u \prec_p v$ denotes that u is the parent node of v. If u and v represent the same node, we have $u = v$, and both $u \preceq_d v$ and $u \preceq_a v$ hold. In the following discussion, we do not differentiate between a node and its label if without ambiguity.

For a given query $Q = \{k_1, k_2, ..., k_m\}$ and an XML document D, we use L_i to denote the inverted Dewey label list of k_i, of which all labels are sorted in document order. Let $LCA(v_1, v_2, ..., v_m)$ be the lowest common ancestor (LCA) of nodes $v_1, v_2, ..., v_m$, the LCAs of Q on D are defined as $LCA(Q) = \{v|v = LCA(v_1, v_2, ..., v_m), v_i \in L_i (1 \leq i \leq m)\}$. E.g., the LCAs of $Q = \{XML, Tom\}$ on D in Fig. 1 are nodes 1.2 and 1.2.2.

In the past few years, researchers have proposed many LCA-based semantics [3, 5, 8, 9, 15, 15], among which SLCA [13, 15] is one of the most widely adopted semantics. Compared with LCA, SLCA defines a subset of $LCA(Q)$, of which no LCA in the subset is the ancestor of any other LCA, which can be formally defined as $SLCASet = SLCA(Q) = \{v|v \in LCA(Q)$ and $\nexists v' \in LCA(Q)$, such that $v \prec_a v'\}$. In Fig. 1, although 1.2 and 1.2.2 are LCAs of $Q = \{XML, Tom\}$, only 1.2.2 is an SLCA node for Q, because 1.2 is an ancestor of 1.2.2.

Based on the set of matched SLCA nodes, there are three kinds of subtree results: (1) *complete subtree (CSubtree)*[5, 15]; (2) *path subtree (PSubtree)* [6]; (3) *matched subtree (MSubtree)*, which is a subtree rooted at v satisfying the constraints of *monotonicity* and *consistency* [7, 11], which can be further interpreted by the changing of data and query, respectively. *Data monotonicity* means that if we add a new node to the data, the number of query results should be (non-strictly) monotonically increasing. *Query monotonicity* means that if we add a keyword to the query, then the number of query results should be (non-strictly) monotonically decreasing. *Data consistency* means that after a data insertion, each additional subtree that becomes (part of) a query result should contain the newly inserted node. *Query consistency* means that if we add a new keyword to the query, then each additional subtree that becomes (part of) a query result should contain at least one match to this keyword. [11] has proved that if all nodes of a subtree t_v satisfy the constraint of "*keywords subsumption*", then t_v must satisfy the constraints of *monotonicity* and *consistency*, that is, t_v is an MSubtree. According to Example 1, we know that compared with CSubtrees and PSubtrees, MSubtrees contain all necessary information after removing nodes that do not satisfy the constraint of keywords subsumption, and are more self-explanatory and compact.

To construct MSubtrees, existing method [11] needs to firstly *scan* all node labels to compute qualified SLCA nodes, then *rescan* all node labels to construct the initial subtrees. After that, they need to buffer these subtrees in memory and apply the constraint of keywords subsumption on each node of these subtrees to prune nodes with keyword sets subsumed by that of their sibling nodes, which is inefficient in time and space.

Considering that an MSubtree may still contain redundant information (discussed in Section 1), in this paper, we focus on efficiently constructing *tightest matched subtree (TMSubtree)* results based on SLCA semantics. Intuitively, a TMSubtree is an MSubtree after removing redundant information. Constructing TMSubtree results based on ELCA semantics [7] is similar and therefore omitted for limited space.

3 Result Enumeration

3.1 Insight into the MaxMatch Algorithm

The MaxMatch algorithm [11] returns MSubtree results that are rooted at SLCA nodes and satisfy the constraint of "keywords subsumption". For a given query $Q = \{k_1, ..., k_m\}$ and an XML document D, supposing that $L_1(L_m)$ is the Dewey label list of occurrence of the least (most) frequent keyword of Q, d is the depth of D. As shown in Algorithm 1, the MaxMatch algorithm works in three steps to produce all MSubtree results.

Step 1 (line 1): MaxMatch finds from the m inverted Dewey label lists the set of SLCA nodes, i.e., $SLCASet$, by calling the IL algorithm [15]. The cost of this step is $O(md|L_1|\log|L_m|)$. In this step, all Dewey labels are processed once.

Step 2 (line 2): MaxMatch calls function $groupMatches$ to construct the set of groups, i.e., $groupSet$. As shown in $groupMatches$, it needs to firstly merge the m lists into a single list with cost $O(\log m \sum_1^m |L_i|)$, then sequentially rescan all labels and insert each one to a certain group (if possible) with cost $O(d \sum_1^m |L_i|)$. Since $d \gg \log|m|$ in practice, the cost of this step is $O(d \sum_1^m |L_i|)$. In this step, all Dewey labels are processed twice to construct the set of groups.

Step 3 (line 3-4): For each group g, MaxMatch firstly constructs the PSubtree, then traverse it to prune redundant information. The overall cost of Step 3 is $O(min\{|D|,\, d\sum_1^m |L_i|\} \cdot 2^m)$, where 2^m is the cost of checking whether the set of distinct keywords of node v is subsumed by that of its sibling nodes, if not, then v is a node that satisfies the constraint of keywords subsumption.

Therefore, the time complexity of Algorithm 1 is $O(max\{min\{|D|, d\sum_1^m |L_i|\} \cdot 2^m\}, md|L_1|\log|L_m|\})$. Moreover, as the MaxMatch algorithm needs to buffer all groups in memory before Step 3, its space complexity is $O(d\sum_1^m |L_i|)$.

3.2 The Tightest Matched Subtree

Definition 1. (Tightest Matched Subtree (TMSubtree)) *For an XML tree D and a keyword query Q, let $S_v \subseteq Q$ be the set of distinct keywords that*

```
Algorithm 1: MaxMatch($Q$)/*$Q = \{k_1, ..., k_m\}$*/

$SLCASet \leftarrow findSLCA(L_1, L_2, ..., L_m)$
$groupSet \leftarrow groupMatches(L_1, L_2, ..., L_m, SLCASet)$
foreach group $g \in groupSet$ do
    $pruneMatches(g)$

Function $groupMatches(L_1, L_2, ..., L_m, SLCASet)$
$L \leftarrow merge(L_1, L_2, ..., L_m)$
sequentially scan each Dewey label $s \in SLCASet$ to construct $groupSet$,
where each group $g_s$ corresponds to an SLCA node $s$
sequentially scan each Dewey label $n \in L$, and put $n$ to group $g_s$ if
$s \in SLCASet$ satisfies that $s \preceq_a n$,
return $groupSet$

Procedure $pruneMatches(group\ g)$
process all Dewey labels of $g$ to construct a PSubtree $t$
foreach node $n$ of $t$ do /*traversing $t$ in depth-first order*/
    if $n$ satisfies the constraint of keywords subsumption then output $n$
```

*appear in the subtree rooted at v, $S_v \neq \emptyset$. A subtree t is a **TMSubtree** iff t's root node is an SLCA node, and each node v of t satisfies that for each sibling node v' of v, $S_v \not\subset S_{v'}$, and for each set of sibling nodes $\{v_1, v_2, ..., v_n\}$ satisfying $S_{v_1} = S_{v_2} = ... = S_{v_n}$, only one of them is kept for presentation.*

Intuitively, a TMSubtree is an MSubtree with redundant information being removed, it can be generated from the corresponding PSubtree by removing all nodes that do not satisfy the constraint of keywords subsumption and just keeping one node for a set of sibling nodes that have the same keyword set.

Definition 2. (Maximum TMSubtree) *Let t be a TMSubtree of Q, v a node of t, $S_v \subseteq Q$ the set of distinct keywords that appear in the subtree rooted at v, $v.level$ the level value of v in t. We say t is a **maximum TMSubtree** if it satisfies the following conditions:*

1. *v has $|S_v|$ child nodes $v_1, v_2, ..., v_{|S_v|}$,*
2. *if $|S_v| \geq 2 \wedge v.level < d$, then for any two child nodes $v_i, v_j (1 \leq i \neq j \leq |S_v|)$, $|S_{v_i}| = |S_{v_j}| = |S_v| - 1 \wedge |S_{v_i} \cap S_{v_j}| = |S_v| - 2$,*
3. *if $|S_v| = 1 \wedge v.level < d$, then v has one child node v_1 and $S_v = S_{v_1}$,*

Lemma 1. *Given a maximum TMSubtree t, t is not a TMSubtree any more after inserting any keyword node into t without increasing t's depth.*

Proof. Suppose that t is not a maximum matched subtree result, then there must exist a non-leaf node v of t, such that we can insert a node v_c into t as a child node of v, v_c satisfies that $S_{v_c} \subseteq S_v$. Obviously, there are four kinds of relationships between S_{v_c} and S_v:

(1) $|S_{v_c}| = |S_v| = 1$. In this case, v has one child node that contains the same keyword as v. Obviously, v_c cannot be inserted into t according to Definition 1.

(2) $S_{v_c} = S_v \wedge |S_v| \geq 2$. Since v has $|S_v|$ child nodes and each one contains $|S_v| - 1$ keywords, for each child node v_i $(1 \leq i \leq |S_v|)$ of v, we have $S_{v_i} \subset S_v = S_{v_c}$. According to Definition 1, all existing child nodes of v should be removed if v_c is inserted into t, thus v_c cannot be inserted into t as a child node of v in such a case.

(3) $|S_{v_c}| = |S_v| - 1 \wedge |S_v| \geq 2$. According to condition 2, all existing child nodes of v contain all possible combinations of keywords in S_v, thus there must exist a child node v_{c_i} of v, such that $S_{v_c} = S_{v_{c_i}}$, which contradicts Definition 1, thus v_c cannot be inserted into t in this case.

(4) if $|S_{v_c}| < |S_v| - 1 \wedge |S_v| \geq 2$. According to condition 2, all existing child nodes of v contain all possible combinations of keywords in S_v, thus there must be a child node v_{c_i} of v, such that $S_{v_c} \subset S_{v_{c_i}}$, which also contradicts Definition 1, thus v_c cannot be inserted into t in such a case.

In summary, if t is a maximum TMSubtree result, no other keyword node can be inserted into t, such that t is still a TMSubtree without increasing t's depth. ■

Theorem 1. *Given a keyword query $Q = \{k_1, k_2, ..., k_m\}$ and one of its TMSubtree t of depth d, if $d \leq m$, then t has at most $2m!$ nodes; otherwise, the number of nodes of t is bounded by $(d - m + 2)m!$.*

Proof. Assume that t is a maximum TMSubtree, obviously, the number of nodes at 1^{st} level of t is $1 = C_m^m$.

For the 2^{nd} level, since $S_{t.root} = Q$, $t.root$ has $|S_{t.root}| = C_m^{m-1}$ child nodes, of which each one contains $|S_{t.root}| - 1$ distinct keywords.

Thus we have Formula 1 to compute the number of nodes at the i^{th} level.

$$N(i) = N(i-1) \cdot C_{m-i+2}^{m-i+1} = P_m^{i-1}, 1 \leq i \leq m. \tag{1}$$

If $d \leq m$, the total number of nodes in t is

$$N = \sum_{i=1}^{m} P_m^{i-1} = m! \sum_{i=1}^{m} \frac{1}{(m-i+1)!} < 2m! \tag{2}$$

If $d > m$, each node at the m^{th} level of t contains only one keyword, and all levels greater than m contains the same number of nodes as that of the m^{th} level. According to Formula 1, we know that $N(m) = m!$, thus the total number of nodes in t is

$$N = \sum_{i=1}^{m} P_m^{i-1} + (d-m)m! < (d-m+2)m! \tag{3}$$

Therefore if $d \leq m$, a TMSubtree t has at most $2m!$ nodes, otherwise, the number of nodes of t is bounded by $(d - m + 2)m!$. ■

Example 2. Given a keyword query $Q = \{k_1, k_2, k_3\}$, Fig. 2 shows three subtree results, according to Definition 1, they are all TMSubtrees. Obviously, by fixing their depths, no other node with any kind of combination of k_1 to k_3 can be inserted into these TMSubtree results according to Lemma 1, that is, they are all maximum TMSubtrees. The TMSubtree in Fig. 2 (A) has 4 nodes. Since its depth is 2 and is less than the number of keywords, i.e., 3, it satisfies Theorem 1 since $4 < 2 \times 3! = 12$. The TMSubtree in Fig. 2 (B) is another maximum TMSubtree with 10 nodes, and also satisfies Theorem 1 since $10 < 2 \times 3! = 12$. Fig. 2 (C) is also a maximum TMSubtree with 16 nodes. Since the depth of the TMSubtree is 4 and is greater than the number of keywords of Q, it still satisfies Theorem 1 since $16 < (4 - 3 + 2) \times 3! = 18$.

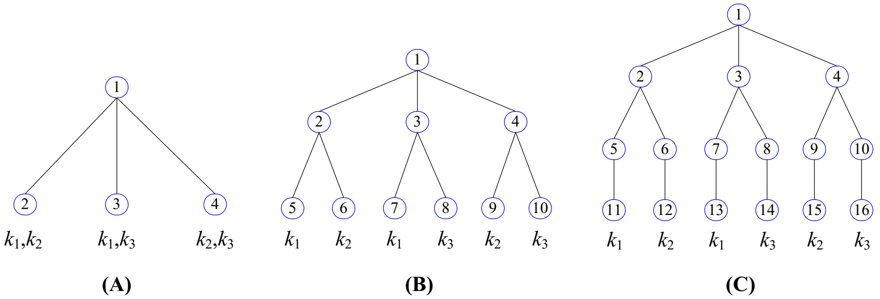


Fig. 2. Illustration of three possible TMSubtrees for keyword query $Q = \{k_1, k_2, k_3\}$

3.3 The Algorithm

Compared with the MaxMatch algorithm that produces all subtree results in three steps, the basic idea of our method is directly constructing all subtree results in the procedure of processing all Dewey labels. The benefits of our method lie in two aspects: (1) the buffered data in memory is largely reduced, (2) each Dewey label is visited only once. The first benefit comes from Theorem 1, which guarantees that our method does not need to buffer huge volumes of data in memory as [11] does; the second benefit is based on our algorithm.

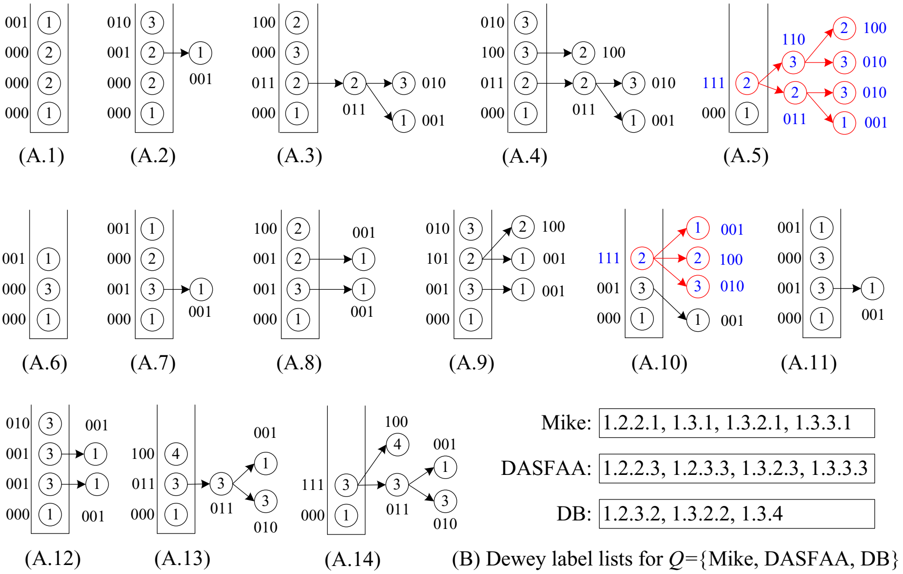
In our algorithm, for a given keyword query $Q = \{k_1, k_2, ..., k_m\}$, each keyword k_i corresponds to a list L_i of Dewey labels sorted in document order, L_i is associated with a cursor C_i pointing to some Dewey label of L_i. C_i can move to the Dewey label (if any) next to it by using $advance(C_i)$. Initially, each cursor C_i points to the first Dewey label of L_i.

As shown in Algorithm 2, our method sequentially scans all Dewey labels in document order. The main procedure is very simple: for all nodes that have not been visited yet, in each iteration, it firstly chooses the currently minimum Dewey label by calling the selectMinLabel function (line 3), then processes it by calling the pushStack procedure (line 4), and finally, it moves C_k forwardly to the next element in L_k (line 5). After all Dewey labels are processed, our algorithm pops all in-stack elements (line 6), then output the last TMSubtree result to terminate the processing (line 7 to 8).

During the processing, our algorithm uses a stack S to temporarily maintain all components of a Dewey label, where each stack element e denotes a component of a Dewey label. e is associated with two variables: the first is a binary bitstring indicating which keyword is contained in the subtree rooted at e; the second is a set of pointers pointing to its child nodes, which is used to maintain intermediate subtrees.

The innovation of our method lies in that our method immediately outputs each TMSubtree result t_v when finding v is a qualified SLCA node, which makes it more efficient in time and space. Specifically, in each iteration (line 2 to 5 of Algorithm 2), our method selects the currently minimum Dewey label C_k in line 3, then pushes all components of C_k into S in line 4. The pushStack procedure firstly pops from S all stack elements that are not the common prefix of C_k and the label represented by the current stack elements, then pushes all C_k's components that are not in the stack into S. To pop out an element from S, the pushStack procedure will call popStack procedure to complete this task. The popStack procedure is a little more tricky. It firstly checks whether the popped element v represents an LCA node. If v is not an LCA node, popStack firstly transfers the value of v's bitstring to its parent node in S (line 12), then insert subtree t_v into $t_{top(S)}$ in line 13. In line 14 to 18, popStack will delete all possible redundant subtrees by checking the subsumption relationship between the keyword set of v and that of its sibling nodes. If v is an LCA node (line 3) and located after the previous LCA node u in document order (line 4), it means that u is an SLCA node if it is not an ancestor of v (line 5), then popStack directly outputs the TMSubtree result rooted at u in line 6. The subtree rooted at u is then deleted (line 7), and u points to v in line 8. If v is an LCA node but located before u, it means that v is not an SLCA node, thus we directly delete the subtree rooted at v (line 10).

Example 3. Consider the XML document D in Fig. 1 and query $Q = \{$Mike, DASFAA, DB$\}$. The inverted Dewey label lists for keywords of Q are shown in Fig. 3 (B). The status of processing these labels are shown in Fig. 3 (A.1) to (A.14). In this example, we use "001" ("010" or "100") to indicate that "Mike" ("DASFAA" or "DB") is contained in a subtree rooted at some node. After 1.2.2.1 is pushed into stack, the status is shown in Fig. 3 (A.1), where the bitstring of the top element of S is "001" indicating that node 1.2.2.1 contains "Mike". The second pushed label is 1.2.2.3, the status is shown in Fig. 3 (A.2). Note that after an element is popped out from the stack, its bitstring is transferred to its parent in the stack. The next two labels are processed similarly. Before 1.3.1 is pushed into the stack, we can see from Fig. 3 (A.5) that the bitstring of the top element in S is "111", which means that node 1.2 contains all keywords. After 1.3.1 is pushed into stack, the subtree rooted at 1.2 is temporarily buffered in memory. As shown in Fig. 3 (A.10), before 1.3.3.1 is pushed into stack, the last component of 1.3.2 will be popped out from the stack. Since the bitstring of the current top element in S is "111" (Fig. 3 (A.10)), we know that 1.3.2 is an LCA node. According to line 5 of popStack procedure, we know that the previous LCA, i.e., 1.2, is an SLCA node, thus we output the matched subtree result rooted at 1.2. After that, the subtree rooted at 1.3.2 will be temporarily buffered in memory. When the last component of 1.3 is popped out from S (Fig. 3 (A.14)), according to line 3 of popStack procedure, we know that 1.3 is an LCA node. According to line 4 of popStack procedure, we know that the previous LCA node, i.e., 1.3.2, is located after 1.3 in document order, thus we immediately know that 1.3 is not an SLCA node, and delete the subtree rooted at 1.3 in line 10 of popStack procedure. Finally, we output the TMSubtree rooted at 1.3.2 in line 7 of Algorithm 2. Therefore for Q, the two TMSubtree results are rooted at 1.2 and 1.3.2, respectively.



(A.1) (A.2) (A.3) (A.4) (A.5)

(A.6) (A.7) (A.8) (A.9) (A.10) (A.11)

(A.12) (A.13) (A.14)

Mike: 1.2.2.1, 1.3.1, 1.3.2.1, 1.3.3.1

DASFAA: 1.2.2.3, 1.2.3.3, 1.3.2.3, 1.3.3.3

DB: 1.2.3.2, 1.3.2.2, 1.3.4

(B) Dewey label lists for Q={Mike, DASFAA, DB}

Fig. 3. Running status for Q ={Mike, DASFAA, DB}

As shown in Algorithm 2, for a given keyword query $Q = \{k_1, k_2, ..., k_m\}$ and an XML document D of depth d, our method just needs to sequentially scan all labels in the m inverted label lists *once*, therefore the overall I/O cost of Algorithm 2 is $O(\sum_1^m |L_i|)$.

Now we analyze the time complexity of our algorithm. Since our algorithm needs to process all components of each involved Dewey label of the given keyword query $Q = \{k_1, k_2, ..., k_m\}$, the total number of components processed in our method is bounded by $d \sum_1^m |L_i|$. During processing, each one of the $d \sum_1^m |L_i|$ components will be inserted into a subtree and deleted from the same subtree just once, and the cost of both inserting and deleting a component is $O(1)$. When inserting a subtree into another subtree, the operation of checking the subsumption relationship between the two keyword sets of two sibling nodes will be executed at most m times according to Definition 2. Therefore, the overall time complexity is $O(dm \sum_1^m |L_i|)$.

```
Algorithm 2: mergeMatching($Q$)            /*$Q = \{k_1, ..., k_m\}$*/

$u \leftarrow 1$                                    /*$u$ is the root node initially*/
while ($\exists i(\neg$ eof($L_i$))) do
    $C_k \leftarrow$ selectMinLabel($Q$)
    pushStack($S, C_k$)                              /*$S$ is the stack*/
    advance($C_k$)
while ($\neg$ isEmpty ($S$)) do popStack($S$)
output the subtree rooted at $u$
delete the subtree rooted at $u$

Function selectMinLabel($Q$)
$c_{min} \leftarrow C_1$
foreach($2 \leq i \leq m$) do
    if($C_i \prec_d c_{min}$) then $c_{min} \leftarrow C_i$
return $c_{min}$

Procedure pushStack($S, C_k$)
$n \leftarrow$ the length of the longest common prefix of $C_k$ and the Dewey label in $S$
while ($|S| > n$) do
    popStack($S$)
foreach($|S| < i \leq |C_k|$) do
    push($S, C_k[i]$)
top($S$).$bit \leftarrow$ top($S$).$bit$ OR 1 << ($k - 1$)      /*bitwise OR operation*/

Procedure popStack($S$)
/*Suppose that $e_1.e_2...e_n$ is a Dewey label of a node, and all the $n$ components
are in the stack $S$. In this procedure, $v$ denotes the last component $e_n$ popped
from $S$ if it is used for bit operation; otherwise, it represents Dewey label
$e_1.e_2...e_n$ or the node itself*/
$flag \leftarrow\sim (\sim 0 << m)$        /*$flag$ is a bitstring with 1 on the right $m$ bits*/
$v \leftarrow$ pop($S$)   /*$v$ denotes the Dewey label consists of all components of $S$ */
if(($v.bit$ AND $flag$) = $flag$) then                /*bitwise AND operation*/
    if($u \prec_d v$) then
        if($u \not\prec_a v$) then
            output the subtree rooted at $u$
        delete the subtree rooted at $u$
        $u \leftarrow v$
    else
        delete the subtree rooted at $v$
else
    $top(S).bit \leftarrow top(S).bit$ OR $v.bit$
    add subtree rooted at $v$ to the subtree rooted at $top(S)$
    foreach(sibling node $v'$ of $v$) do
        if(($v'.bit$ AND $v.bit$) = $v.bit$) then
            delete the subtree rooted at $v$
        else if(($v'.bit$ AND $v.bit$) = $v'.bit$) then
            delete the subtree rooted at $v'$

Function eof($L_i$)
if (all Dewey labels of $L_i$ are processed) then return TRUE
else return FALSE
```

Since our method is executed in pipelined way, at any time, it just needs to maintain at most d subtrees, where each one is smaller than a maximum TMSubtree. According to Theorem 1, each matched subtree result contains at most $max\{2m!, (d - m + 2)m!\}$ nodes. Therefore, the space complexity of our method is $O(d \cdot max\{2m!, (d - m + 2)m!\})$. Since d and m are very small in practice, the size of these subtrees buffered in memory is very small.

Note that to output the name of nodes in a TMSubtree result, existing methods may either store all path information in advance by suffering from huge storage space [3], or use the extended Dewey labels [12] by affording additional cost on computing the name of each node according to predefined rules. In contrast, our method maintains another hash mapping between each path ID and the path information, the total number of index entries is the number of nodes in the dataguide index [4] of the XML tree, which is very small in practice. To derive for each node its name on a path, we maintain in each Dewey label a path ID after the last component, thus we can get the name of each node on a path in constant time.

4 Experimental Evaluation

4.1 Experimental Setup

Our experiments were implemented on a PC with Intel(R) Core(TM) i5 M460 2.53 GHz CPU, 2 GB memory, 500 GB IDE hard disk, and Windows XP professional as the operating system.

The algorithm used for comparison is the MaxMatch algorithm [11][1], which was implemented based on the Stack [15], IL [15] and IMS [13] algorithms to test the impacts of different algorithms on the overall performance, and is denoted as MaxMatch-Stack, MaxMatch-IL and MaxMatch-IMS, respectively. All these algorithms and our mergeMatching algorithm were implemented using Microsoft VC++. All results are the average time by executing each algorithm 10 times on hot cache.

We use XMark[2] dataset for our experiments because it possesses complex schema, which can test the features of different algorithms in a more comprehensive way. The size of the dataset is 582MB, it contains 8.35 million nodes, the maximum depth and the average depth of the XML tree are 12 and 5.5 respectively.

We have selected 30 keywords, which are classified into three categories according to their frequencies: (1) low frequency (100-1000), (2) median frequency (10000-40000), and (3) high frequency (300000-600000). Based on these keywords, we generated four groups of queries as shown in Table 1: (1) four queries (Q1 to Q4) with 2, 3, 4, 5 keywords of low frequency; (2) four queries (Q5 to Q8) of median frequency; (3) four queries (Q9 to Q12) of high frequency; (4) 20 queries (Q13 to Q32) with keywords of random frequency.

[1] The MaxMatch algorithm is used to output TMSubtrees in our experiment.

[2] `http://monetdb.cwi.nl/xml`

Table 1. Queries used in our experiment

Query	Keywords	Query	Keywords
Q1	villages,hooks	Q17	baboon,patients
Q2	baboon,patients,arizona	Q18	tissue,shocks,order
Q3	cabbage,tissue,shocks,baboon	Q19	province,bold,increase
Q4	shocks,necklace,cognition,cabbage,tissue	Q20	cabbage,male,female
Q5	female,order	Q21	listitem,emph,arizona
Q6	privacy,check,male	Q22	patients,school,gender
Q7	takano,province,school,gender	Q23	patients,school,gender,text
Q8	school,gender,education,takano,province	Q24	bold,increase,hooks,takano
Q9	bold,increase	Q25	male,female,keyword,incategory
Q10	date,listitem,emph	Q26	emph,arizona,villages,education
Q11	incategory,text,bidder,date	Q27	check,bidder,date,baboon
Q12	bidder,date,keyword,incategory,text	Q28	school,gender,time,baboon,patients
Q13	text,tissue	Q29	tissue,shocks,order,province,bold
Q14	takano,province	Q30	female,keyword,incategory,cabbage,male
Q15	incategory,cabbage	Q31	arizona,villages,education,listitem,emph
Q16	check,bidder	Q32	bidder,date,necklace,cognition,check

4.2 Performance Comparison and Analysis

Fig. 4 (A), (B), (C) and Fig. 5 show the results of applying the four algorithms to queries with keywords of low, median, high and random frequencies. From these figures we know that our method is more efficient than the MaxMatch algorithm for all queries. The reason lies in that no matter which algorithm, i.e., either Stack, IL or IMS, is adopted for SLCA computation, it cannot avoid the following operations: (1) MaxMatch needs to re-scan all Dewey labels at least twice in line 2 of Algorithm 1 to construct the set of groups (see Section 3.1 for detailed reasons), (2) MaxMatch needs to firstly construct all PSubtrees, then make pruning by traversing all nodes of each PSubtree. On the contrary, our method only needs to scan all Dewey labels once, which makes it very efficient compared with the MaxMatch algorithm.

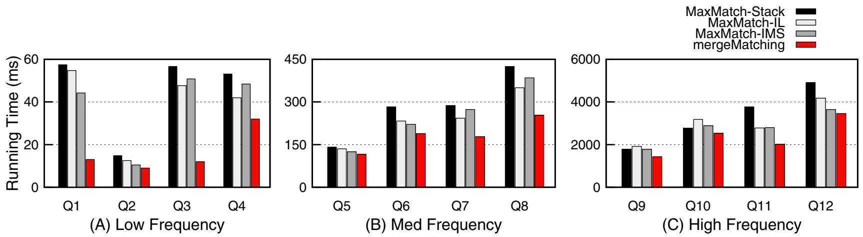


Fig. 4. Running time for queries with keywords of low, median and high frequencies

The second observation from Fig. 4 (A), (B), (C) and Fig. 5 is that for a given keyword query, different algorithms on SLCA computation will impose comparatively less influence on the overall performance of the MaxMatch algorithm when generating TMSubtree results, this is because the running time used to constructe PSubtrees and make pruning on these PSubtrees usually occupies a majority of the total running time to generate all TMSubtree results.

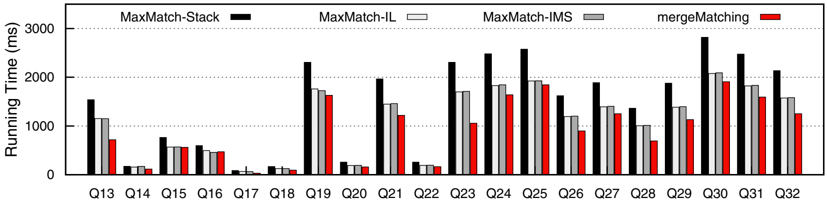


Fig. 5. Running time for queries with keywords of random frequencies

Besides, we show in Fig. 6 the scalability when executing Q9 on XMark datasets with different sizes (from 116MB to 1745MB (15x)). The query time of the MaxMatch-Stack, MaxMatch-IL, MaxMatch-IMS and our mergeMatching algorithms grow sublinearly with the increase of the data size. Also, mergeMatching consistently saves about 24.5%, 34.1%, 24.6% time when compared with MaxMatch-Stack, MaxMatch-IL and MaxMatch-IMS, respectively. For other queries, we have similar results, which are omitted due to space limit.

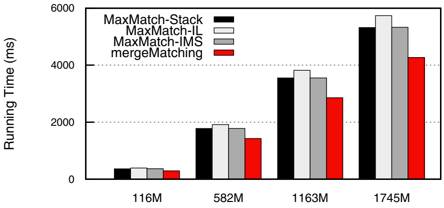


Fig. 6. Running time of Q9 on XML documents of different sizes

5 Conclusions

Considering that TMSubtree is more self-explanatory and compact than CSubtree and PSubtree, but existing methods on subtree result computation need to re-scan all Dewey labels more than once, in this paper, we focus on efficient construction of *TMSubtree* results for keyword queries on XML data based on SLCA semantics. We firstly proved the upper bound for the size of a given TMSubtree, that is, it has at most $2m!$ nodes if $d \leq m$; otherwise, its size is bounded by $(d - m + 2)m!$, where d is the depth of a given TMSubtree, and m is the number of keywords of the given query Q. Then we proposed a pipelined algorithm to accelerate the computation of TMSubtree results, which only needs to sequentially scan all Dewey labels *once* without buffering huge volumes of intermediate results, because the space complexity of our method is $O(d \cdot max\{2m!, (d - m + 2)m!\})$, and in practice, d and m are very small, the size of the buffered subtrees is very small. The experimental results in Section 4 verify the benefits of our algorithm in aiding keyword search over XML data.

Acknowledgment. This research was partially supported by grants from the Natural Science Foundation of China (No. 61073060, 61040023, 61103139) and the Fundamental Research Funds of Hebei Province (No. 10963527D). Zhifeng Bao was partially supported by the Singapore Ministry of Education Grant No. R252-000-394-112 under the project name of UTab.

References

1. Bao, Z., Ling, T.W., Chen, B., Lu, J.: Effective xml keyword search with relevance oriented ranking. In: ICDE, pp. 517–528 (2009)
2. Chen, L.J., Papakonstantinou, Y.: Supporting top-k keyword search in xml databases. In: ICDE, pp. 689–700 (2010)
3. Cohen, S., Mamou, J., Kanza, Y., Sagiv, Y.: Xsearch: A semantic search engine for xml. In: VLDB, pp. 45–56 (2003)
4. Goldman, R., Widom, J.: Dataguides: Enabling query formulation and optimization in semistructured databases. In: VLDB, pp. 436–445 (1997)
5. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: Xrank: Ranked keyword search over xml documents. In: SIGMOD Conference, pp. 16–27 (2003)
6. Hristidis, V., Koudas, N., Papakonstantinou, Y., Srivastava, D.: Keyword proximity search in xml trees. IEEE Trans. Knowl. Data Eng. 18(4), 525–539 (2006)
7. Kong, L., Gilleron, R., Lemay, A.: Retrieving meaningful relaxed tightest fragments for xml keyword search. In: EDBT, pp. 815–826 (2009)
8. Li, G., Feng, J., Wang, J., Zhou, L.: Effective keyword search for valuable lcas over xml documents. In: CIKM, pp. 31–40 (2007)
9. Li, Y., Yu, C., Jagadish, H.V.: Schema-free xquery. In: VLDB, pp. 72–83 (2004)
10. Liu, Z., Chen, Y.: Identifying meaningful return information for xml keyword search. In: SIGMOD Conference, pp. 329–340 (2007)
11. Liu, Z., Chen, Y.: Reasoning and identifying relevant matches for xml keyword search. PVLDB 1(1), 921–932 (2008)
12. Lu, J., Ling, T.W., Chan, C.Y., Chen, T.: From region encoding to extended dewey: On efficient processing of xml twig pattern matching. In: VLDB, pp. 193–204 (2005)
13. Sun, C., Chan, C.Y., Goenka, A.K.: Multiway slca-based keyword search in xml data. In: WWW, pp. 1043–1052 (2007)
14. Tatarinov, I., Viglas, S., Beyer, K.S., Shanmugasundaram, J., Shekita, E.J., Zhang, C.: Storing and querying ordered xml using a relational database system. In: SIGMOD Conference, pp. 204–215 (2002)
15. Xu, Y., Papakonstantinou, Y.: Efficient keyword search for smallest lcas in xml databases. In: SIGMOD Conference, pp. 537–538 (2005)
16. Xu, Y., Papakonstantinou, Y.: Efficient lca based keyword search in xml data. In: EDBT, pp. 535–546 (2008)
17. Zhou, R., Liu, C., Li, J.: Fast elca computation for keyword queries on xml data. In: EDBT, pp. 549–560 (2010)

Stars on Steroids: Fast Evaluation of Multi-source Star Twig Queries in RDBMS

Erwin Leonardi[1,2], Sourav S. Bhowmick[1,2], and Fengrong Li[3]

[1] Singapore-MIT Alliance, Nanyang Technological University, Singapore
[2] School of Computer Engineering, Nanyang Technological University, Singapore
[3] Japan Advanced Institute of Science and Technology, Japan
{lerwin,assourav}@ntu.edu.sg, lifr@nagoya-u.jp

Abstract. Despite a large body of work on XML twig query processing in relational environment, systematic study of XML join evaluation has received little attention in the literature. In this paper, we propose a novel and non-traditional technique for fast evaluation of *multi-source star twig* queries in a *path materialization*-based RDBMS. A *multi-source star twig* joins different XML documents on values in their nodes and the *XQuery graph* takes a star-shaped structure. Such queries are prevalent in several domains such as life sciences. Rather than following the conventional approach of generating one huge complex SQL query from a twig query, we translate a star query into a list of SQL sub-queries that only materializes *minimal information* of underlying XML subtrees as intermediate results. Experiments carried out confirm that our proposed approach build on top of an off-the-shelf commercial RDBMS has excellent real-world performance.

1 Introduction

Efficient evaluation of XML queries that correlate (join) multiple input documents to integrate data from different sources is highly important due to its several real-world applications. For example, querying biological data across multiple sources is a key activity for many biologists. If these sources represent data in XML format (e.g., INTERPRO (www.ebi.ac.uk/interpro/), UNIPROT (www.expasy.ch/sprot/), PDB (www.pdb.org), EMBL (www.ebi.ac.uk/embl/)), then XQuery can be used to formulate meaningful queries over these data sources. Figure 1 shows three example queries. Observe that Q_1, Q_2, and Q_3 correlate four, three, and two data sources, respectively. Also, in each query the join conditions share a common data source. For instance, in Q_1 UNIPROT is joined with INTERPRO, PDB, and EMBL. Similarly, in Q_2 UNIPROT is joined with INTERPRO and EMBL. Consequently, each of these queries can be represented as a star-shaped query graph where a node represents a data source and an edge represents existence of a join expression between a pair of sources. We refer to such queries as *multi-source star twig* queries (*star queries* for brevity). *In this paper, we focus on fast evaluation of this type of queries in a relational environment.*

At first glance, it may seem that we can efficiently evaluate star queries by leveraging on an existing relational XQuery processor, *c.f.*, [8,12] and relying on its query optimization capabilities. Specifically in an XQuery processor, an XQuery query is often rewritten to an equivalent, logically simpler XQuery and then translated to a *single*, complex SQL query, *c.f.*, [8]. Optimization of an XQuery query is achieved in two

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 110–125, 2012.

© Springer-Verlag Berlin Heidelberg 2012

QID	Query	# of Results

Q1	declare namespace PDBx = 'http://deposit.pdb.org/pdbML/pdbx.xsd';
	for $entry in fn:collection('UNIPROT')/uniprot/entry,
	    $interpro in fn:collection('INTERPRO')/interprodb/interpro,
	    $embl in fn:collection('EMBL')/EMBL_Services/entry,
	    $pdb in fn:collection("PDB")/PDBx:datablock
	let $ref2PDB := $entry/dbReference[@type="PDB"]/@id
	let $ref2EMBL := $entry/dbReference[@type="EMBL"]/@id
	let $ref2InterPro := $entry/dbReference[@type="InterPro"]/@id
	let $temp:=$embl/@created
	where $entry/keyword = 'ATP-binding'
	    and $entry/organism/name = 'Human'
	    and $interpro/pub_list/publication/journal = 'Science'
	    and fn:starts-with(xs:string($temp), '1996')
	    and $pdb/PDBx:citationCategory/PDBx:citation/PDBx:country = "US"
	    and $pdb/PDBx:citationCategory/PDBx:citation/PDBx:year = "1997"
	    and $pdb/PDBx:cellCategory/PDBx:cell/@entry_id = $ref2PDB
	    and $interpro/@id = $ref2InterPro
	   and $embl/@accession= $ref2EMBL
	return $entry/reference/citation/title;
		64

Q2	for $entry in fn:collection('UNIPROT')/uniprot/entry,
	    $interpro in fn:collection('INTERPRO')/interprodb/interpro,
	    $embl in fn:collection('EMBL')/EMBL_Services/entry
	let $ref2EMBL := $entry/dbReference[@type="EMBL"]/@id
	let $ref2InterPro := $entry/dbReference[@type="InterPro"]/@id
	let $temp:=$embl/@created
	where $entry/keyword = 'ATP-binding'
	  and $entry/organism/name = 'Human'
	  and $interpro/pub_list/publication/journal = 'Science'
	  and fn:starts-with(xs:string($temp), '1996')
	  and $interpro/@id = $ref2InterPro and $embl/@accession= $ref2EMBL
	return $entry/name;
		4

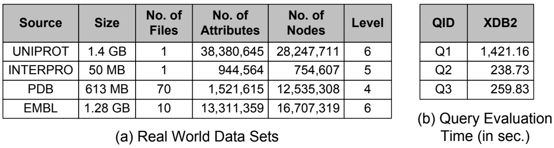
Q3	for $entry in fn:collection('UNIPROT')/uniprot/entry,
	   $embl in fn:collection('EMBL')/EMBL_Services/entry
	let $ref2EMBL := $entry/dbReference[@type="EMBL"]/@id
	let $temp:=$embl/@created
	where $entry/keyword = 'ATP-binding'
	      and $entry/organism/name = 'Human'
	      and fn:starts-with(xs:string($temp), '1996')
	      and $embl/@accession= $ref2EMBL
	return $entry/gene;
		11

Fig. 1. Examples of star twig queries

stages. Logical query optimization (sometimes also called query rewrite) [8, 12, 13] results in rewrites of XQuery statements to avoid duplicate and full navigations. On the other hand, physical query optimization depends on the storage method of the data being queried. For instance, we can store and query XML representations of INTERPRO, UNIPROT, PDB, and EMBL using XML support provided by DB2.

Unfortunately, query performance still remains a bottleneck. To get a better understanding of this problem, we experimented with the datasets in Figure 2(a) and queries $Q_1 – Q_3$. Figure 2(b) shows the query evaluation times in DB2. Observe that it can take from 4 minutes to more than 20 minutes to evaluate these queries. *Is it possible to design a scheme that can address this performance bottleneck?* In this paper, we demonstrate that techniques build on top of an existing off-the-shelf RDBMS can make up for a large part of the limitation. In particular, we show that the above queries can be evaluated in *less than a minute*.

We take an alternative non-traditional strategy that bypasses logical XQuery optimization and relies solely on the relational optimizer to achieve superior performance for evaluating star queries. This approach is perhaps surprising because the design goals of our strategy seem to be diametrically opposite to traditional relational XQuery processors. Specifically, given a star query Q, our proposed algorithm translates it into a list of SQL queries without undertaking any logical query optimization over a *path materialization*-based storage scheme [6]. First, SQL queries for materializing the *identifiers* of



Source	Size	No. of Files	No. of Attributes	No. of Nodes	Level
UNIPROT	1.4 GB	1	38,380,645	28,247,711	6
INTERPRO	50 MB	1	944,564	754,607	5
PDB	613 MB	70	1,521,615	12,535,308	4
EMBL	1.28 GB	10	13,311,359	16,707,319	6

(a) Real World Data Sets

QID	XDB2
Q1	1,421.16
Q2	238.73
Q3	259.83

(b) Query Evaluation Time (in sec.)

Fig. 2. Dataset and query evaluation times in DB2

nodes or subtrees satisfying the expressions in the `return` clause are generated. Based on these materialized identifiers, SQL queries for *non-join* expressions in the `where` clause are generated followed by queries for *join* expressions. These queries are executed in sequence and the results are materialized in temporary tables. The identifiers of nodes (subtrees) satisfying Q are then computed from these materialized results. A key feature of these materialized results is that we only store *minimal* information (identifiers of nodes) required for evaluating Q. This obviously has positive impact on the storage and query processing costs of temporary tables as we can efficiently store large intermediate result nodes for a given query. Finally, the last step of the algorithm is to issue an SQL query to retrieve *complete* information from the base table(s) containing XML documents by matching the identifiers of the result subtrees.

Our proposed approach has excellent performance. It is significantly faster than XML support of DB2 $v9.5$ (highest observed factor being 158 times), which relies on conventional XQuery optimization techniques. Somewhat unexpectedly, we shall also show that the proposed technique outperforms one of the fastest XQuery processor (MONETDB/XQuery [2]) for several queries (highest observed factor being 46 times)!

The rest of our paper is organized as follows. We compare our approach with related work in Section 2. Section 3 formally defines the notion of multi-source star twig queries. Section 4 presents in detail the algorithm for evaluating star queries on top of a path materialization-based relational storage. We evaluate and compare the performance of our proposed technique through an extensive set of experiments in Section 5. Section 6 concludes the paper and suggests future work.

2 Related Work

There is a wealth of work on evaluating XPath expressions in a tree-unaware RDBMS [6, 7, 14] and tree-aware environment [2, 6]. However, these efforts mainly focus on various XPath axes and not on XML join operation. In all these efforts, the SQL translation algorithms generate a single complex SQL whereas here we focus on generating a sequence of SQL queries. Consequently, in this paper we materialize *minimal* subtree information to reduce the size of the intermediate tables generated by the list of SQL queries. Complete information related to subtrees that satisfy the query is only retrieved during the final step of query execution. Such "lazy" approach to retrieve subtree information is not necessary in approaches that are based on a single SQL query. Also, in contrast to previous efforts, the proposed algorithm is sensitive to the order of evaluation of different components (i.e., `return` clause, *join* expressions, *non-join* expressions) of the star queries.

There has been efforts related to translating XML queries to SQL in XML publishing environment [9]. In XPeranto [16], an XQuery query is transformed into an XML Query Graph Model (XQGM) and composed with the view definition. Then it is translated to a single "outer union" SQL query to be evaluated inside the relational engine. The Agora [11] project uses *local-as-view* (LAV) approach to translate the XML query into a SQL query over virtual relational schema and then rewriting this SQL query into a query over the real relational schema. MARS [4] uses both local-as-view and *global-as-view* (GAV) approaches. It first compiles the queries, views and constraints from XML into the relational framework and then determines all minimal reformulations of the relational queries under the relational integrity constraints using a cost-based approach. In contrast, our approach is build on top of XML storage framework and translates a specific type of XML query to *a list of* SQL queries instead of a single SQL query.

More germane to this work is efforts in the XML publishing environment that translate an XML query to a list of SQL queries [5]. In [5], mapping from the relational schema to the XML view is specified using a declarative query language RXL. In order to create the XML view, optimal set of SQL queries are generated to extract and group data from the underlying relational engine. In general, there are $2^{|E|}$ possible translations of an RXL query into one or more queries, where $|E|$ is the number of edges in the query's *view tree* (representation that makes it clear how to generate queries). In contrast, the number of SQL queries in our approach is linear to the number data sources to be joined and the number of *output expressions* in the query.

3 Multi-source Star Twig Pattern

3.1 Multi-source Twig Pattern

Most XML processors, both native and relational, have overwhelmingly focused on *single-source* twig queries modeled as a twig pattern tree [6]. A *single-source* twig query is evaluated on a set of documents represented by a single XML schema or DTD. However, as discussed in Section 1, related data in many real-world applications may span across multiple data sources with different schemas. Consequently, our query model should support queries over such multiple data sources using joins. We refer to such twig queries as *multi-source twig patterns*.

A multi-source twig pattern Q is a graph with three types of nodes: location step query node (QNode), logical-AND node (ANode), and return node (RNode). Each Q has a single node of type RNode which represents the output node. While the label of ANode is always "AND", QNodes' and RNodes' labels are tags. An edge in Q can be of two types, namely, *axes edge* and *join edge*. The former represents parent-child or attribute relationship[1] between a pair of nodes belonging to the same source whereas the latter connects two nodes from two different sources. Specifically, a join edge (q_1, q_2) asserts that q_1 and q_2 have equal value[2]. We distinguish the RNode by underlined tag; and axes and join edges as direct and dashed edges, respectively.

[1] We consider XPath navigation only along the `child` (`/`) and `attribute` (`/@`) axes. Extension to other navigation axis is orthogonal to the proposed technique.

[2] We currently support equality join condition but inequality join condition can be supported easily.

Observe that a multi-source twig query can be represented by an XQuery query $Q = (\mathcal{F}, \mathcal{L}, \mathcal{W}, \mathcal{R})$ where $\mathcal{F}$ is a set of `for` clause items, $\mathcal{L}$ is a set of expressions defined using the `let` clause, $\mathcal{W}$ is a set of predicates in the `where` clause, and $\mathcal{R}$ is an output expression specified in the `return` clause. Specifically, the syntax of Q is as follows.

```
   FOR   $\$x_1\ in\ p_1, \ldots, \$x_n\ in\ p_n$
   LET   $\$y := q_1$
   LET   $\ldots$
 WHERE   $b_1 \wedge b_2 \wedge \ldots \wedge b_k \wedge c_1 \wedge c_2 \wedge \ldots \wedge c_m$
RETURN   $r$
```

Note that there must be at least two `for` clause items in Q that are bound to two different document sources. The `let` clause simply declares a variable and gives it a value. We categorize the *where-expressions* in $\mathcal{W}$ into two types, namely *join expressions* and *non-join expressions*. A *join expression* involves predicates that express join conditions over two different document sources. On the other hand, a *non-join expression* expresses a filtering condition on a single source. Note that a join expression can also be expressed in a `for` clause using qualifier. In this paper, we ignore join expressions in the `for` clause, which can always be reformulated away using `where` clause. Finally, an *output expression* r in the `return` clause is of type `RNode`.

Definition 1. ***[XQuery Representation of Multi-source Twig]*** *Let var be the name of variable binding, exp be a path expression, op $\in \{=, \neq, >, \geq, <, \leq\}$ be an operator, and val be a value. Given an expression exp, the function $source(exp)$ maps exp to the document source D over which exp is valid. Then, an* ***XQuery*** *query $Q = (\mathcal{F}, \mathcal{L}, \mathcal{W}, \mathcal{R})$ is a multi-source twig query if the followings are true.*

- *$\mathcal{F}$ is a set of* `for` *clause items such that $|\mathcal{F}| \geq 2$. An item $f \in \mathcal{F}$ is a triple (var, dsName, exp), where $source(exp) = dsName$. Furthermore, $\exists\ f_i \in \mathcal{F} \wedge f_j \in \mathcal{F}$ such that $f_i.dsName \neq f_j.dsName$ for $i \neq j$ and $1 < i, j \leq |\mathcal{F}|$.*
- *$\mathcal{L}$ is a set of* `let` *clause items where $l \in \mathcal{L}$ is a 2-tuple (var, exp).*
- *Let S and T be path expressions containing $var = f.var$ or $var = l.var$ where $f \in \mathcal{F}$, $l \in \mathcal{L}$. Then, $\mathcal{W}$ is a set of conjunctive predicates in the* `where` *clause where $\mathcal{W} = \mathcal{J} \cup \mathcal{C}$ and $\mathcal{J} \cap \mathcal{C} = \emptyset$. $\mathcal{J}$ is a non-empty set of join expressions where $b \in \mathcal{J}$ is of the form S op T. $\mathcal{C}$ is a set of non-join expressions where $c \in \mathcal{C}$ is of the form S op val.*
- *$\mathcal{R}$ is the* `return` *clause containing output expression r, which is a 2-tuple (var, exp) where $var = f.var$ or $var = l.var$, $f \in \mathcal{F}$, and $l \in \mathcal{L}$.* □

3.2 Star Twig Pattern

An XQuery representation of a multi-source twig query can be conveniently represented using an *XQuery graph*. Similar to a query graph of an SQL query, an XQuery graph is an undirected graph with nodes $D_1 \ldots D_n$. For every join expression between the document sources D_i and D_j, we add an edge between D_i and D_j. This edge is labeled by the join expression. The nodes are labeled with corresponding non-join expressions.

An XQuery graphs can have many different shapes such as chain queries, star queries, tree queries, cyclic queries, clique queries, etc. Note that these classes are not disjoint

Algorithm 1: The ***StarTwig2SQL*** algorithm.

```
   Input: Star twig query $Q$
   Output: A list of SQL queries $SQLList$
1  Initialize $SQLList = \emptyset$;
2  $(\mathcal{F}, \mathcal{W}, \mathcal{R}) \leftarrow$ parseXQuery($Q$) /* Phase 1*/;
3  $SQLList$.add(outputExp2SQL($\mathcal{R}$)) /* Phase 2 */;
4  $(\mathcal{J}, \mathcal{C}) \leftarrow$ distinguishExp($W$) /* Phase 3 */;
5  $SQLList$.add(whereExp2SQL($\mathcal{F}, \mathcal{J}, \mathcal{C}, \mathcal{R}$));
6  $SQLList$.add(finalResultQueryGen($\mathcal{R}$)) /* Phase 4 */ ;
7  return $SQLList$
```

and that some classes are subsets of other classes. In this paper, *we focus on star queries joining different* XML *documents*. Intuitively, in a *multi-source star twig* query all join expressions share a common document source and hence forms a star-shaped query graph. For example, queries in Figure 1 are examples of star twig queries. Formally, it is defined as follows.

Definition 2. *[Multi-source Star Twig Query] Let $Q = (\mathcal{F}, \mathcal{L}, \mathcal{W}, \mathcal{R})$ be a multi-source XQuery query. Then Q is called a* ***multi-source star twig*** *query if any one of the following conditions is true: (a) $|\mathcal{J}| = 1$ and $source(b.S) \neq source(b.T)$ where $b \in \mathcal{J}$. (b) If $|\mathcal{J}| > 1$ then $\forall\, i \neq j$ $source(b_i.S) = source(b_j.S)$ and $source(b_i.S) \neq source(b_i.T)$ where $b_i \in \mathcal{J}$, $b_j \in \mathcal{J}$ and $1 \leq i, j \leq |\mathcal{J}|$.* □

4 Star Twig Query Evaluation

In this section, we shall elaborate on the algorithm for translating a star twig query to a list of SQL queries over relational framework. State-of-the-art relational approaches for XML storage can be broadly classified into four types, namely, *node* approach, *edge* approach, *path materialization (*PM*)* approach, and DTD approach [6]. For the sake of generality, in this paper we assume that the XML data are schemaless. Since the PM approach has advantages over the rest when XML data are schemaless [6], our proposed algorithm is built on top of this storage approach. Importantly, we present a generic algorithm that is independent of any specific PM approach. We assume that paths, contents of leaf nodes, and attributes associated with a XML tree are materialized in Paths, PathsContent, and Attributes relations, respectively. The reader may refer to [10] for an example of how various subroutines in the algorithm can be realized on a specific PM approach.

The algorithm for SQL translation is shown in Algorithm 1. The algorithm consists of four phases as discussed below.

Phase 1: XQuery Parsing. In the first phase, a multi-source star twig query Q is parsed using XPath 2.0/XQuery 1.0 Parser Build [1] (Line 02). During the parsing process, the algorithm identifies different components of Q based on the star twig query model discussed in the preceding section. Also, the algorithm replaces the variable references in Q with the expressions defined in the `let` clause (if any). The output

Algorithm 2: The ***outputExp2SQL*** algorithm.

```
Input: An output expression $r \in \mathcal{R}$
Output: An SQL query $\_SQL$
Initialize $\_SQL = \infty$;
if ($r$ is an attribute node) then
    $PathExp \leftarrow$ pathExpOfParentNode($r$);
else
    $PathExp \leftarrow r.absExp$;
$PathIDs \leftarrow$ getAllPathID($PathExp$);
$Level \leftarrow$ getNodeLevel($PathExp$);
$Source = r.dsName$;
$\_SQL$.genSQL($PathIDs$, $Level$, $Source$);
return $\_SQL$
```

Algorithm 3: The ***whereExp2SQL*** algorithm.

```
Input: $\mathcal{F}$, $\mathcal{J}$, $\mathcal{C}$, $r \in \mathcal{R}$
Output: A list of SQL queries $SQLList$
Initialize $SQLList = \emptyset$;
for (each $f \in \mathcal{F}$) do
    $\mathcal{C}_f \leftarrow$ getNonJoinExp($f.var$, $\mathcal{C}$);
    if ($f.var = r.var$) then
        $SQL \leftarrow$ translateWhereNonJoin($r$, $f$, $\mathcal{C}_f$);
    else
        $SQL \leftarrow$ translateWhereJoin($r$, $f$, $\mathcal{C}_f$, $\mathcal{J}$);
    $SQL \leftarrow$ INSERT INTO $T\_$" $+\mathcal{R}+$ "_"$+\mathcal{F}$.indexOf($f$)$+$ " " $+SQL$;
    $SQLList$.add($SQL$);
return $SQLList$
```

of this phase are a set of `for` clause items $\mathcal{F}$, a set of *where-expressions* $\mathcal{W}$, and the output expression $r \in \mathcal{R}$. In addition, we also determine the absolute path expressions of $r \in \mathcal{R}$, $c \in \mathcal{C}$, and $b \in \mathcal{J}$. The absolute path expression of r is denoted by $r.absExp$. For example, consider $r =$ (*$entry*, "`/name`") in Q_2 (Figure 1). Then $r.absExp$ is "`/uniprot/entry/name`" as *$entry* is bound to the expression "`/uniprot/entry`".

Phase 2: OutputExp2SQL Translation. In this phase, the algorithm analyzes the output expression $r \in \mathcal{R}$ and generates an SQL query for materializing the *identifiers* of the XML subtrees that satisfy r (Line 03). An *identifier* of a node n in an XML tree D (denoted by nId) is one or more attributes of n that can uniquely identify n in D. The materialized identifiers of r are stored in a temporary relation $\mathsf{PathU}(\mathsf{DocId}, \mathsf{nId})$. Note that we materialize the identifiers instead of entire subtrees because it is more space-efficient (the size of materialized identifier table is always smaller than or equal to the table containing entire materialized subtrees). Also, we do not need to materialize the

Algorithm 4: The ***translateWhereNonJoin*** algorithm.

```
Input: An output expression $r$, a for clause item $f$, $\mathcal{C}_f$
Output: An SQL query $SQL$
Initialize $selectClause$, $fromClause$, $whereClause$, $optionClause$;
$dataS \leftarrow$ source($r.var$);
for ($i = 1$ to $|\mathcal{C}_f|$) do
    $c = \mathcal{C}_f[i]$;
    if (c is a condition on attribute) then
        Generate SQL statements for $fromClause$ and $whereClause$;
    else
        Add SQL statements to $whereClause$;
    Add instance of PathsContent representing $dataS$ to the $fromClause$;
    if ($i > 1$) then
        $whereClause$.add(evalTwig($c.absExp$, $\mathcal{C}_f[i-1].absExp$));
Add instances of PathsContent to $fromClause$;
$whereClause$.add(evalTwig($r.absExp$, $c.absExp$));
Add nId, docId to $selectClause$;
$SQL = selectClause + fromClause + whereClause$;
return $SQL$
```

level of r *explicitly* as it can be computed on-the-fly in a PM-based storage approach. It is worth mentioning that the identifier scheme is not tightly coupled to any specific numbering scheme as any scheme that can uniquely identify nodes in an XML tree can be used as an identifier. For instance, the *preorder* and *dewey order* values of nodes can be used for *region encoding* and *dewey number-based* labeling schemes, respectively [6].

Given an output expression $r \in \mathcal{R}$, the *OutputExp2SQL* algorithm depicted in Algorithm 2 works as follows. First, the algorithm determines whether r involves an attribute node (Line 02). If it does, then the algorithm retrieves the absolute path expression of its parent node (Line 03). Otherwise, the absolute path expression of r is used (Line 05). This expression is stored in the variable *PathExp*. Based on *PathExp*, a set of path ids is retrieved from the Paths table (Line 06). Also, the algorithm computes the node level of r using *PathExp*. Then the SQL query for materializing nodes satisfying r (PathU table) is generated by exploiting the Paths, Attributes, and PathsContent relations.

Phase 3: WhereExp2SQL Translation. Here, we translate the *where-expression* into a list of SQL queries. The result of each SQL query is stored in a temporary table that is an instance of the relation TempTable(DocId, nId). This phase starts by distinguishing the join and non-join expressions followed by invocation of the *WhereExp2SQL* algorithm (Lines 04–05, Algorithm 1). Intuitively, for each pair of output expression r and an item f of the for clause expressions it generates an SQL query. If r and f refer to the *same* data source D then it generates a non-join query that evaluates the conditions specified in the *where-expression* related to D. Otherwise, if r and f refer to *different* sources, namely D_1 and D_2, respectively, then a join query is generated that satisfies the join predicate(s) as well as non-join predicates on D_2. For example, there are three pairs of

Algorithm 5: The ***translateWhereJoin*** algorithm.

```
Input: An output node $r$, a for clause item $f$, $\mathcal{C}_f$, $\mathcal{J}$
Output: An SQL query $SQL$
Initialize $selectClause$, $fromClause$, $whereClause$, $optionClause$;
processExpressions($\mathcal{C}_f$);
$i = |\mathcal{C}_f| + 1$;
$\mathcal{J}_f \leftarrow \mathcal{J}$.getJoinExp($f$);
$\mathcal{J}_r \leftarrow \mathcal{J}$.getJoinExp($r$);
if ($\mathcal{J}_f \cap \mathcal{J}_r = \emptyset$) then
    processJoinExp($\mathcal{J}_f$.getS(), $\mathcal{J}_f$.getT(), $i$, $\mathcal{C}_f[|\mathcal{C}_f|].absExp$);
    processJoinExp($\mathcal{J}_r$.getS(), $\mathcal{J}_r$.getT(), $i$, $\mathcal{C}_f[|\mathcal{C}_f|].absExp$);
else
    $\mathcal{J}_x = \mathcal{J}_f \cap \mathcal{J}_r$;
    processJoinExp($\mathcal{J}_x$.getS(), $\mathcal{J}_x$.getT(), $i$, $\mathcal{C}_f[|\mathcal{C}_f|].absExp$);
$i = i + 1$;
Add instances of PathsContent relation to $fromClause$;
$whereClause$.add(evalTwig($r.absExp$, $T.absExp$));
Add nId, docId to $selectClause$;
$SQL = selectClause + fromClause + whereClause$;
return $SQL$
```

(r, f) in Q_2, namely (*$entry*, *$entry*), (*$entry*, *$interpro*), and (*$entry*, *$embl*). Since (*$entry*, *$entry*) refers to the same data source (UNIPROT), the algorithm generates an SQL query that retrieves those nodes in the PathU table (generated by Phase 2) that satisfy the non-join predicates on UNIPROT. The results of this query is stored in an instance of TempTable (denoted by T_1). On the other hand, data sources of (*$entry*, *$interpro*) are not identical and hence a join query is generated that selects nodes from PathU that satisfy the join predicate (Line 11 in Q_2) and the conditions on INTERPRO (Line 9). The results of this query is stored in the temporary table T_2.

The *WhereExp2SQL* algorithm is depicted in Algorithm 3. For each $f \in \mathcal{F}$, first, it retrieves $\mathcal{C}_f \subseteq \mathcal{C}$, where $\forall c \in \mathcal{C}_f$ $c.var = f.var$ (Line 03). Then, it determines whether r and f are bound to the same data source. If $r.var = f.var$, then join across data sources is not necessary. In this case, the algorithm will invoke the *translateWhereNonJoin* algorithm (Line 05). Otherwise, it invokes the *translateWhereJoin* algorithm (Line 07). The generated SQL query is stored in a variable called *SQLList*. Lastly, an `insert` statement is appended to the generated SQL query so that the results of the query can be directly stored in the temporary table. We now elaborate on the *translateWhereNonJoin* and *translateWhereJoin* procedures.

The *translateWhereNonJoin* Algorithm. Given a pair of (r, f) representing the *same* source, the *translateWhereNonJoin* algorithm (Algorithm 4) generates a non-join SQL query. For each *where-expression* $c \in \mathcal{C}_f$, the algorithm first checks whether c is specified on an attribute. If it is, then it will add SQL statements to the `where` and `from` clauses of the translated SQL query by exploiting the Paths and Attributes relations (Line 06). These statements retrieve path ids based on $c.absExp$ satisfying the value conditions on the attributes. If c is *not* specified on an attribute then these expressions are added to the where clause (Line 08). If there are more than one conditions in $\mathcal{C}_f$, then it represents a twig query pattern. Consequently, SQL statement for evaluating the twig pattern is added using *evalTwig* procedure (Line 10). Next, the algorithm specifies the condition between these expressions and r using *evalTwig* procedure (Line 11) as we are interested in only those nodes that satisfy the output expression. The PathU table is used for this purpose. The SQL query generated by the *translateWhereExp* algorithm returns the identifiers of nodes satisfying r that satisfy expressions in $\mathcal{C}_f$.

The *translateWhereJoin* Algorithm. Given a pair of (r, f) representing two *different* sources, the *translateWhereJoin* algorithm (Algorithm 5) generates the join query. First, the SQL fragment for evaluation of non-join conditions on the source represented by f is generated as we are interested in those joinable nodes that satisfy the predicates on this source. The steps for this are encapsulated in the *processExpression* function and are same as the ones in Lines 03–11 of Algorithm 4. The next step is to general the SQL fragment for the join expressions (Lines 04–11). First, the algorithm creates two subsets of $\mathcal{J}$, namely $\mathcal{J}_f$ and $\mathcal{J}_r$, containing sets of join expressions involving the sources of f and r, respectively (Lines 04–05). If $(\mathcal{J}_f \cap \mathcal{J}_r = \emptyset)$, then the algorithm processes each of the join expressions by invoking the *processJoinExp* algorithm twice (Lines 07–10). The functions **getS** and **getT** return the S and T components of a join expression $S\ op\ T$, respectively (see Definition 1). Let us elaborate on this scenario with an example. Consider a query Q that contains $f.var \in \{f_1, f_2, f_3\}$. Let $\mathcal{J}$ in Q contains two join expressions, namely $S_1 = T$ and $S_2 = T$ where S_1, S_2, and T are path expressions representing three different data sources and contain $f_2.var$, $f_3.var$, and $f_1.var$, respectively. Let $\mathcal{R} = \{r\}$ where $r.var = f_3.var$. Now consider the pair (r, f_2) in the context of Algorithm 5. Here $\mathcal{J}_f = \{\text{“}S_1 = T\text{”}\}$ and $\mathcal{J}_r = \{\text{“}S_2 = T\text{”}\}$. Since $\mathcal{J}_f \cap \mathcal{J}_r = \emptyset$, Lines 07–08 are executed. In this case, the algorithm processes the join between S_1 and T first followed by the join between S_2 and T. Note that there is no join expression of the form “$S_1 = S_2$”. On the other hand, if $(\mathcal{J}_f \cap \mathcal{J}_r \neq \emptyset)$, then the algorithm will retrieve the common join expressions (denoted by $\mathcal{J}_x$) between $\mathcal{J}_f$ and $\mathcal{J}_r$, and process them by invoking the *processJoinExp* procedure (Lines 11). To elaborate further, consider the pair (r, f_1) in the context of the above example. Here $\mathcal{J}_f = \mathcal{J}_r = \{\text{“}S_1 = T\text{”}\}$. Hence, Line 11 is executed. The objective of *processJoinExp* procedure is to generate the SQL fragments involving the join expressions. For each join expression $S\ op\ T$, it checks the type of node (attribute or element) in S and T and corresponding SQL fragments are added to where and from clauses. Lastly, Algorithm 5 evaluates the twig fragment consisting of the join expression and the output expression r using *evaluateTwig* procedure. Note that this procedure is similar to the one discussed in the context of *TranslateWhereNonJoin* algorithm.

Phase 4: Final Results Generator. Finally, this phase generates a set of SQL queries for retrieving the final results in two steps. The first step is to combine the results of SQL queries generated in Phase 3 (Line 02). Note that the results of these queries can be combined by performing intersection operation over them. The results of the SQL queries generated in this step are sets of *identifiers* satisfying the output expression r stored in the PathUFinal(DocId, nId) table. In the second step the algorithm retrieves *complete*

QID	Query	# of Results

Q4	for $entry in fn:collection('UNIPROT')/uniprot/entry,
	    $interpro in fn:collection('INTERPRO')/interprodb/interpro
	let $ref2Interpro := $entry/dbReference[@type="InterPro"]/@id
	where $entry/keyword = 'Vision' and $entry/organism/name = 'Human'
	  and $interpro/pub_list/publication/journal = "Nature"
	  and $interpro/@id = $ref2Interpro
	return $entry/gene;
		31

Q5	for $entry in fn:collection('UNIPROT')/uniprot/entry,
	    $interpro in fn:collection('INTERPRO')/interprodb/interpro
	let $ref2Interpro := $entry/dbReference[@type="InterPro"]/@id
	where $entry/keyword = "Vision" and $entry/organism/name = "Human"
	  and $interpro/pub_list/publication/journal = "Nature"
	  and $interpro/pub_list/publication/year = "1990"
	  and $interpro/@id = $ref2Interpro
	return $entry/gene;
		13

Q6	declare namespace PDBx='http://deposit.pdb.org/pdbML/pdbx.xsd';
	for $entry in fn:collection('UNIPROT')/uniprot/entry,
	    $pdb in fn:collection('PDB')/PDBx:datablock
	let $ref2PDB := $entry/dbReference[@type="PDB"]/@id
	where $entry/keyword = '3D-structure'
	  and $entry/organism/name = 'Human'
	  and $pdb/PDBx:citationCategory/PDBx:citation/PDBx:year = "2005"
	  and $pdb/PDBx:cellCategory/PDBx:cell/@entry_id = $ref2PDB
	return $entry/sequence;
		2

Q7	declare namespace PDBx='http://deposit.pdb.org/pdbML/pdbx.xsd'; for $entry in fn:collection("UNIPROT")/uniprot/entry, $interpro in fn:collection("INTERPRO")/interprodb/interpro, $pdb in fn:collection("PDB")/PDBx:datablock let $ref2PDB := $entry/dbReference[@type="PDB"]/@id let $ref2Interpro := $entry/dbReference[@type="InterPro"]/@id where $entry/organism/name="Human" and $interpro/pub_list/publication/journal = "Nature" and $interpro/pub_list/publication/year = "2000" and $pdb/PDBx:citationCategory/PDBx:citation/PDBx:country = "UK" and $pdb/PDBx:citationCategory/PDBx:citation/PDBx:year = "2002" and $interpro/@id = $ref2Interpro and $pdb/PDBx:cellCategory/PDBx:cell/@entry_id = $ref2PDB return $entry/gene;	2

Q8	declare namespace PDBx='http://deposit.pdb.org/pdbML/pdbx.xsd';
	for $entry in fn:collection('UNIPROT')/uniprot/entry,
	    $interpro in fn:collection('INTERPRO')/interprodb/interpro,
	    $pdb in fn:collection('PDB')/PDBx:datablock
	let $ref2PDB := $entry/dbReference[@type="PDB"]/@id
	let $ref2Interpro := $entry/dbReference[@type="InterPro"]/@id
	where $entry/organism/name="Mouse"
	  and $interpro/pub_list/publication/journal = "Nature"
	  and $interpro/@id = $ref2Interpro
	  and $pdb/PDBx:citationCategory/PDBx:citation/PDBx:country = "US"
	  and $pdb/PDBx:cellCategory/PDBx:cell/@entry_id = $ref2PDB
	return $entry;
		1

Fig. 3. Query set

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
SX	35.62	24.47	16.48	18.23	17.94	3.34	18.46	19.26
XDB2	1,421.16	238.73	259.83	130.80	131.35	128.08	112.77	104.73

Fig. 4. Query evaluation times (in sec.)

information related to these nodes (remaining attributes in PathsContent) for generating the final result. Specifically, it generates an SQL query by joining the PathUFinal and PathsContent tables. The results are sorted in document order.

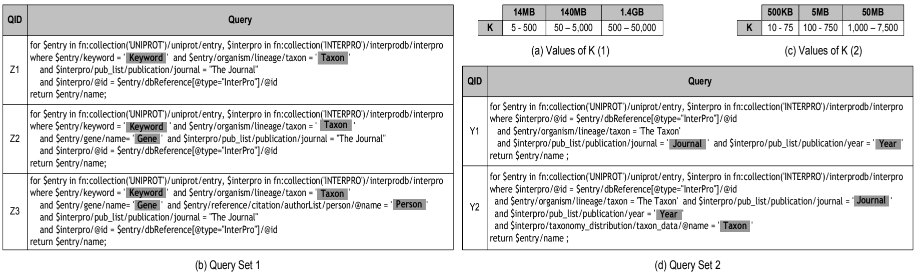
Theorem 1. *Let $Q = (\mathcal{F}, \mathcal{L}, \mathcal{W}, \mathcal{R})$ be a multi-source star twig query involving n different data sources. Then, the total number of* SQL *queries generated from Q is $(n + k)$ where (a) if the output expression $r \in \mathcal{R}$ does not contain attribute node then $k = 3$; (b) Otherwise, $k = 4$.* □

Due to space constraints, the proof is given in [10].

5 Experimental Results

Prototype for star query evaluation system was implemented on top of a PM-based XML database system called SUCXENT++ [14] (denoted by SX) using Java JDK 1.6. The experiments were conducted on an Intel machine with Core2 Duo E6550 2.33GHz processor and 3.25GB RAM. The operating system was Windows XP Professional SP3. The RDBMS used was MS SQL Server 2005 Developer Edition.

We compare our approach to the native XML support of IBM DB2 $v9.5$ (denoted by XDB2). XML support of IBM DB2 is also used as performance benchmark in [7]. For SX and XDB2, appropriate indexes were created [10]. Prior to our experiments, we ensure that statistics had been collected. The bufferpool of the RDBMS was cleared before each run. The queries in SX were executed in the *reconstruct* mode where not only the internal nodes are selected, but also all descendants of those nodes. Each query was executed 6 times and the results from the first run were always discarded. For XDB2, we use the *db2batch* Benchmark Tool provided by the system. All rows were fetched from the answer set; however, they were not sent to output.



QID	Query
Z1	for $entry in fn:collection('UNIPROT')/uniprot/entry, $interpro in fn:collection('INTERPRO')/interprodb/interpro where $entry/keyword = 'Keyword' and $entry/organism/lineage/taxon = 'Taxon' and $interpro/pub_list/publication/journal = "The Journal" and $interpro/@id = $entry/dbReference[@type="InterPro"]/@id return $entry/name;
Z2	for $entry in fn:collection('UNIPROT')/uniprot/entry, $interpro in fn:collection('INTERPRO')/interprodb/interpro where $entry/keyword = 'Keyword' and $entry/organism/lineage/taxon = 'Taxon' and $entry/gene/name= 'Gene' and $interpro/pub_list/publication/journal = "The Journal" and $interpro/@id = $entry/dbReference[@type="InterPro"]/@id return $entry/name;
Z3	for $entry in fn:collection('UNIPROT')/uniprot/entry, $interpro in fn:collection('INTERPRO')/interprodb/interpro where $entry/keyword = 'Keyword' and $entry/organism/lineage/taxon = 'Taxon' and $entry/gene/name= 'Gene' and $entry/reference/citation/authorList/person/@name = 'Person' and $interpro/pub_list/publication/journal = "The Journal" and $interpro/@id = $entry/dbReference[@type="InterPro"]/@id return $entry/name;

(b) Query Set 1

	14MB	140MB	1.4GB
K	5 - 500	50 – 5,000	500 – 50,000

(a) Values of K (1)

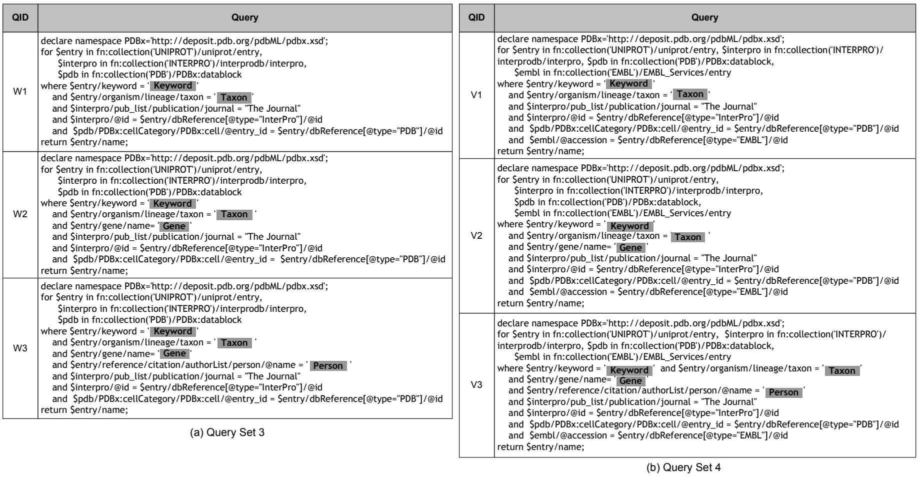
	500KB	5MB	50MB
K	10 - 75	100 - 750	1,000 – 7,500

(c) Values of K (2)

QID	Query
Y1	for $entry in fn:collection('UNIPROT')/uniprot/entry, $interpro in fn:collection('INTERPRO')/interprodb/interpro where $interpro/@id = $entry/dbReference[@type="InterPro"]/@id and $entry/organism/lineage/taxon = 'The Taxon' and $interpro/pub_list/publication/journal = 'Journal' and $interpro/pub_list/publication/year = 'Year' return $entry/name ;
Y2	for $entry in fn:collection('UNIPROT')/uniprot/entry, $interpro in fn:collection('INTERPRO')/interprodb/interpro where $interpro/@id = $entry/dbReference[@type="InterPro"]/@id and $entry/organism/lineage/taxon = 'The Taxon' and $interpro/pub_list/publication/journal = 'Journal' and $interpro/pub_list/publication/year = 'Year' and $interpro/taxonomy_distribution/taxon_data/@name = 'Taxon' return $entry/name ;

(d) Query Set 2

Fig. 5. Synthetic query sets and the K parameter



QID	Query
W1	declare namespace PDBx='http://deposit.pdb.org/pdbML/pdbx.xsd'; for $entry in fn:collection('UNIPROT')/uniprot/entry, $interpro in fn:collection('INTERPRO')/interprodb/interpro, $pdb in fn:collection('PDB')/PDBx:datablock where $entry/keyword = 'Keyword' and $entry/organism/lineage/taxon = 'Taxon' and $interpro/pub_list/publication/journal = "The Journal" and $interpro/@id = $entry/dbReference[@type="InterPro"]/@id and $pdb/PDBx:cellCategory/PDBx:cell/@entry_id = $entry/dbReference[@type="PDB"]/@id return $entry/name;
W2	declare namespace PDBx='http://deposit.pdb.org/pdbML/pdbx.xsd'; for $entry in fn:collection('UNIPROT')/uniprot/entry, $interpro in fn:collection('INTERPRO')/interprodb/interpro, $pdb in fn:collection('PDB')/PDBx:datablock where $entry/keyword = 'Keyword' and $entry/organism/lineage/taxon = 'Taxon' and $entry/gene/name= 'Gene' and $interpro/pub_list/publication/journal = "The Journal" and $interpro/@id = $entry/dbReference[@type="InterPro"]/@id and $pdb/PDBx:cellCategory/PDBx:cell/@entry_id = $entry/dbReference[@type="PDB"]/@id return $entry/name;
W3	declare namespace PDBx='http://deposit.pdb.org/pdbML/pdbx.xsd'; for $entry in fn:collection('UNIPROT')/uniprot/entry, $interpro in fn:collection('INTERPRO')/interprodb/interpro, $pdb in fn:collection('PDB')/PDBx:datablock where $entry/keyword = 'Keyword' and $entry/organism/lineage/taxon = 'Taxon' and $entry/gene/name= 'Gene' and $entry/reference/citation/authorList/person/@name = 'Person' and $interpro/pub_list/publication/journal = "The Journal" and $interpro/@id = $entry/dbReference[@type="InterPro"]/@id and $pdb/PDBx:cellCategory/PDBx:cell/@entry_id = $entry/dbReference[@type="PDB"]/@id return $entry/name;

(a) Query Set 3

QID	Query
V1	declare namespace PDBx='http://deposit.pdb.org/pdbML/pdbx.xsd'; for $entry in fn:collection('UNIPROT')/uniprot/entry, $interpro in fn:collection('INTERPRO')/ interprodb/interpro, $pdb in fn:collection('PDB')/PDBx:datablock, $embl in fn:collection('EMBL')/EMBL_Services/entry where $entry/keyword = 'Keyword' and $entry/organism/lineage/taxon = 'Taxon' and $interpro/pub_list/publication/journal = "The Journal" and $interpro/@id = $entry/dbReference[@type="InterPro"]/@id and $pdb/PDBx:cellCategory/PDBx:cell/@entry_id = $entry/dbReference[@type="PDB"]/@id and $embl/@accession = $entry/dbReference[@type="EMBL"]/@id return $entry/name;
V2	declare namespace PDBx='http://deposit.pdb.org/pdbML/pdbx.xsd'; for $entry in fn:collection('UNIPROT')/uniprot/entry, $interpro in fn:collection('INTERPRO')/interprodb/interpro, $pdb in fn:collection('PDB')/PDBx:datablock, $embl in fn:collection('EMBL')/EMBL_Services/entry where $entry/keyword = 'Keyword' and $entry/organism/lineage/taxon = 'Taxon' and $entry/gene/name= 'Gene' and $interpro/pub_list/publication/journal = "The Journal" and $interpro/@id = $entry/dbReference[@type="InterPro"]/@id and $pdb/PDBx:cellCategory/PDBx:cell/@entry_id = $entry/dbReference[@type="PDB"]/@id and $embl/@accession = $entry/dbReference[@type="EMBL"]/@id return $entry/name;
V3	declare namespace PDBx='http://deposit.pdb.org/pdbML/pdbx.xsd'; for $entry in fn:collection('UNIPROT')/uniprot/entry, $interpro in fn:collection('INTERPRO')/ interprodb/interpro, $pdb in fn:collection('PDB')/PDBx:datablock, $embl in fn:collection('EMBL')/EMBL_Services/entry where $entry/keyword = 'Keyword' and $entry/organism/lineage/taxon = 'Taxon' and $entry/gene/name= 'Gene' and $entry/reference/citation/authorList/person/@name = 'Person' and $interpro/pub_list/publication/journal = "The Journal" and $interpro/@id = $entry/dbReference[@type="InterPro"]/@id and $pdb/PDBx:cellCategory/PDBx:cell/@entry_id = $entry/dbReference[@type="PDB"]/@id and $embl/@accession = $entry/dbReference[@type="EMBL"]/@id return $entry/name;

(b) Query Set 4

Fig. 6. Synthetic query sets

We would also like to observe how "far off" our approach is from one of the fastest XQuery processor (MONETDB/XQuery [2]). Hence, we used the Windows version of MONETDB/XQuery 0.24.0 (denoted as MX) downloaded from `monetdb.cwi.nl/XQuery/Download/index.html` (Win32 builds) for our study.

5.1 Query Evaluation Times on Real Datasets

In our experiments, we used real datasets from life sciences domain as star twig queries are prevalent in this domain. Specifically, we use the XML representations of UNIPROT, PDB, INTERPRO, and EMBL downloaded from their official websites. The features of these datasets are given in Figure 2(a). We chose eight multi-source star twig queries as shown in Figures 1 and 3 that join up to four data sources, and have between three to nine expressions in the `where` clause. We transform these queries to our model (Section 3) if necessary. Observe that the queries are highly selective (small result size).

	Z1				Z2				Z3			
	K=5	K=50	K=250	K=500	K=5	K=50	K=250	K=500	K=5	K=50	K=250	K=500
SX	0.52	0.53	0.54	0.58	0.52	0.54	0.58	0.57	0.78	0.79	0.81	0.83
XDB2	1.95	1.75	2.71	3.03	1.51	1.63	2.33	2.61	2.54	2.75	3.44	3.72
MX	7.47	7.52	7.89	8.33	7.48	7.56	8.08	8.53	7.56	7.89	9.38	10.61
MX-R	0.06	0.09	0.11	1.39	0.06	0.08	0.09	1.30	0.13	0.11	0.14	0.16

(a) UNIPROT (14 MB)

	Z1				Z2				Z3			
	K=50	K=500	K=2,500	K=5,000	K=50	K=500	K=2,500	K=5,000	K=50	K=500	K=2,500	K=5,000
SX	2.38	2.45	2.46	2.49	2.69	2.66	2.72	2.76	4.75	4.82	4.99	4.99
XDB2	13.92	18.59	201.98	393.95	12.92	16.79	40.38	67.18	40.10	45.51	56.00	72.37
MX	GDKmallocmax Error											
MX-R	0.25	11.25	59.39	114.50	0.27	11.53	59.45	114.66	0.27	11.69	61.00	117.17

(b) UNIPROT (140 MB)

	Z1				Z2				Z3			
	K=500	K=5,000	K=25,000	K=50,000	K=500	K=5,000	K=25,000	K=50,000	K=500	K=5,000	K=25,000	K=50,000
SX	27.96	28.20	46.76	47.90	19.17	18.97	19.99	40.04	45.06	45.99	63.45	65.94
XDB2	137.31	175.24	542.73	DNF	138.27	608.10	521.33	DNF	724.47	740.36	1,106.22	DNF

(c) UNIPROT (1400 MB)

	Y1			Y2		
	K=10	K=50	K=75	K=10	K=50	K=75
SX	1.87	1.86	1.87	1.88	1.90	1.91
XDB2	13.15	15.68	18.22	12.94	15.87	15.46
MX	0.80	0.78	0.78	0.80	0.80	0.80
MX-R	1.83	7.48	10.81	2.55	10.39	10.69

(d) INTERPRO (500KB)

	Y1			Y2		
	K=100	K=500	K=750	K=100	K=500	K=750
SX	2.60	2.74	2.80	2.70	2.76	2.78
XDB2	19.82	48.77	62.89	19.47	48.50	49.84
MX	0.89	0.90	0.89	0.89	0.92	0.91
MX-R	18.25	GDKMallocmax Error		22.20	GDKMallocmax Error	

(e) INTERPRO (5MB)

	Y1			Y2		
	K=1,000	K=5,000	K=7,500	K=1,000	K=5,000	K=7,500
SX	5.72	6.40	6.83	6.01	6.74	6.85
XDB2	87.77	381.34	564.97	87.20	383.47	462.89
MX	1.34	1.41	1.42	1.40	1.44	1.48
MX-R	GDKmallocmax Error					

(f) INTERPRO (50MB)

Fig. 7. Query evaluation times (in sec.)

Figure 4 depicts the query evaluation times of SX and XDB2. Note that we did not show any results of MX as it is vulnerable to the virtual memory fragmentation in Windows environment. Consequently, it failed to shred UNIPROT XML (1.4GB in size). Observe that SX significantly outperforms XDB2 for all queries.

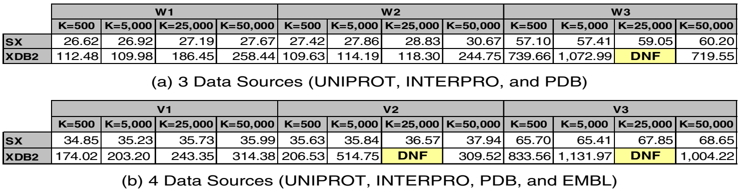
5.2 Query Evaluation Times on Synthetic Datasets

The main objective here is to study the effects of the size of intermediate results on the query evaluation times. In the sequel, the symbol DNF means that the query evaluation did not finish in 30 mins. We compare SX, XDB2, and MX. Note that due to the *GDKmallocmax* error in MX for some queries, we rewrote all queries in MX into *sequential* ones[3]. In sequential queries, non-join expressions are specified as qualifiers in path expressions of `for` clause items instead of specifying them in the `where` clause. In the sequel, we denote the MONETDB system with the rewritten queries as MX-R. We use UNIPROT and INTERPRO datasets and modified them (discussed below) so that the size of intermediate results can be controlled. We set UNIPROT as the output data source.

Varying Intermediate Results of UNIPROT. We vary the size of UNIPROT documents from 14MB to 1.4GB and fix the size of INTERPRO dataset to 50MB. We control the intermediate result size by varying the number of subtrees (denoted as K) that matches a non-join twig query in the XML document(s). The variation of K for different dataset sizes is depicted Figure 5(a). Figure 5(b) depicts the query set used in this set of experiments. These queries are chosen by varying the number of predicates on UNIPROT dataset from 2 to 4. We vary the result size of the highlighted predicates in the `where` clause. For instance, in Z_1 we vary the number of subtrees (K) returned by the following non-join twig condition: `$entry/keyword =‘Keyword’ and $entry/organism/lineage/taxon =‘Taxon’`.

Figures 7(a)–(c) show the query evaluation times. Note that we do not compare MX and MX-R in Figure 7(c) as it is vulnerable to the virtual memory fragmentation. We can make the following observations. Firstly, the cost of query evaluation increases

[3] This error occurred because the system cannot allocate certain amount of memory specified in the error message.



	W1				W2				W3			
	K=500	K=5,000	K=25,000	K=50,000	K=500	K=5,000	K=25,000	K=50,000	K=500	K=5,000	K=25,000	K=50,000
SX	26.62	26.92	27.19	27.67	27.42	27.86	28.83	30.67	57.10	57.41	59.05	60.20
XDB2	112.48	109.98	186.45	258.44	109.63	114.19	118.30	244.75	739.66	1,072.99	DNF	719.55

(a) 3 Data Sources (UNIPROT, INTERPRO, and PDB)

	V1				V2				V3			
	K=500	K=5,000	K=25,000	K=50,000	K=500	K=5,000	K=25,000	K=50,000	K=500	K=5,000	K=25,000	K=50,000
SX	34.85	35.23	35.73	35.99	35.63	35.84	36.57	37.94	65.70	65.41	67.85	68.65
XDB2	174.02	203.20	243.35	314.38	206.53	514.75	DNF	309.52	833.56	1,131.97	DNF	1,004.22

(b) 4 Data Sources (UNIPROT, INTERPRO, PDB, and EMBL)

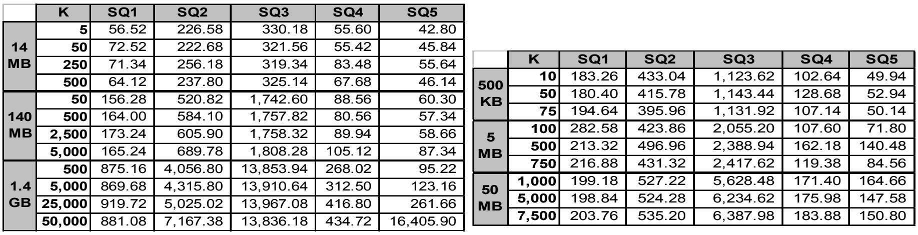
Fig. 8. Query evaluation times (1.4GB UNIPROT (in sec.))

with the size of intermediate results for all approaches. Secondly, SX performs better than XDB2 for all queries. For instance, SX is 158 times faster than XDB2 for Z_1 when $K = 5,000$ (Figure 7(b)). Thirdly, for certain queries SX is faster than MONETDB! It is faster than MX for all queries for 14MB dataset (highest observed factor being 14.8 times). On the other hand, MX-R is faster than SX for 13 out of 24 queries (highest observed factor being 17.9 times). Interestingly, SX outperforms MX-R for remaining queries (up to 46 times faster). We also observe that rewriting the queries to sequential ones in MONETDB performs better than MX and it can evaluate queries that previously cannot be evaluated by MX.

Varying Intermediate Results of INTERPRO. We now fix the UNIPROT dataset size to 140MB and vary the INTERPRO document sizes from 500KB to 50MB. The values of K for this set of experiments are depicted Figure 5(c). Figure 5(d) presents the query set. The numbers of predicates on INTERPRO dataset are set to 2 and 3 for Y_1 and Y_2, respectively. Figures 7(d)–(f) depict the query evaluation times. Similar to above results, SX is faster than XDB2 for all queries (highest observed factor being 82.7 times). However, MX performs better than SX for all queries (up to 4.8 times faster). Interestingly, we observe that MX-R cannot evaluate 10 out of 18 queries because of *GDKmallocmax* error. For the remaining queries, SX outperforms MX-R for 7 out of 8 queries (highest observed factor being 8.2 times). Hence, it is evident that rewriting XQueries to sequential ones in MONETDB may not always be a beneficial strategy.

Varying Number of Data Sources. Next, we vary the number of data sources involved in joins. Note that this also varies the number of sub-queries generated during the evaluation (Theorem 1). In addition, we also vary the intermediate result size of nodes (subtrees) of UNIPROT satisfying output expressions as depicted in Figure 5(a). We used query sets shown in Figures 6(a) and (b) joining three and four data sources, respectively. Figure 8 shows the evaluation times of queries in Figures 6(a) and (b). Note that we do not compare MX and MX-R in Figure 8 due to virtual memory fragmentation problem. Notice that SX is faster than XDB2 for all queries. Furthermore, the number of data sources involved in the join influences the query evaluation time in all approaches.

Evaluation Times of Sub-queries. The above results confirm the strengths of our approach. We now explore further the reasons behind such superior performance by investigating the contributions made by individual sub-queries to the execution costs of the translated SQL queries. We chose Z_2 and Y_2 as our test queries. The translated SQL query of Z_2 and Y_2 each consists of five sub-queries (denoted as SQ_1 to SQ_5). SQ_1



	K	SQ1	SQ2	SQ3	SQ4	SQ5
14 MB	5	56.52	226.58	330.18	55.60	42.80
	50	72.52	222.68	321.56	55.42	45.84
	250	71.34	256.18	319.34	83.48	55.64
	500	64.12	237.80	325.14	67.68	46.14
140 MB	50	156.28	520.82	1,742.60	88.56	60.30
	500	164.00	584.10	1,757.82	80.56	57.34
	2,500	173.24	605.90	1,758.32	89.94	58.66
	5,000	165.24	689.78	1,808.28	105.12	87.34
1.4 GB	500	875.16	4,056.80	13,853.94	268.02	95.22
	5,000	869.68	4,315.80	13,910.64	312.50	123.16
	25,000	919.72	5,025.02	13,967.08	416.80	261.66
	50,000	881.08	7,167.38	13,836.18	434.72	16,405.90

(a) Query Z_2

	K	SQ1	SQ2	SQ3	SQ4	SQ5
500 KB	10	183.26	433.04	1,123.62	102.64	49.94
	50	180.40	415.78	1,143.44	128.68	52.94
	75	194.64	395.96	1,131.92	107.14	50.14
5 MB	100	282.58	423.86	2,055.20	107.60	71.80
	500	213.32	496.96	2,388.94	162.18	140.48
	750	216.88	431.32	2,417.62	119.38	84.56
50 MB	1,000	199.18	527.22	5,628.48	171.40	164.66
	5,000	198.84	524.28	6,234.62	175.98	147.58
	7,500	203.76	535.20	6,387.98	183.88	150.80

(b) Query Y_2

Fig. 9. Sub-queries evaluation times of Z_2 and Y_2 (in msec)

is used to fetch the identifiers of the output nodes (Phase 1). SQ_2 and SQ_3 materialize the results for non-join and join expressions (Phase 3). The PathUFinal relation is generated by SQ_4. SQ_5 retrieves the complete subtrees including the necessary attributes for reconstruction and all the descendant node if the output node is an internal node. We evaluate the evaluation time of each sub-query using SX as shown in Figure 9. Observe that relatively the most expensive query is SQ_3 for both cases. However, the evaluation time is still below $15s$ (significantly lower than the evaluation times of XDB2). On the other hand, SQ_1, SQ_2, SQ_4, and SQ_5 are highly efficient for almost all cases. This is primarily due to (a) efficient support of twig pattern evaluation in a PM-based XML storage approach, (b) space-efficient storage of intermediate results of the queries, and (c) small queries are less likely to stress the query optimizer.

6 Conclusions and Future Work

In this paper, we take a non-traditional approach in evaluating multi-source star twig queries on top of a path-based tree-unaware XML database. Rather than generating one huge complex SQL query, we translate a star query into a list of SQL queries. This is surprising, because when only one SQL query is generated, it has the greatest potential for optimization by the RDBMS. We showed that by materializing only minimal information of underlying XML subtrees as intermediate results we can "turbo-charge" star query processing. Though not elaborated in this paper, it is easy to see that our approach is also applicable to a host of XML databases using relational backend as well as wide varieties of complex XML queries. Our results showed that our proposed technique has excellent real-world performance, outperforming XML join support of DB2 for many queries. Although MONETDB/XQuery [2] is one of the fastest XQuery processor, surprisingly, our results show that our scheme outperforms it for several queries. As part of future work, we would like to extend our approach to larger subset of XML queries.

References

1. W3C. XQuery 1.0 Grammar Test Page (2005), http://www.w3.org/2005/qt-applets/xqueryApplet.html
2. Boncz, P., Grust, T., et al.: MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. In: SIGMOD (2006)
3. Brantner, M., Kanne, C.-C., Moerkotte, G.: Let a Single FLWOR Bloom (to improve XQuery plan generation). In: XSym Workshop (2007)
4. Deutsch, A., Tannen, V.: MARS: A System for Publishing XML from Mixed and Redundant Storage. In: VLDB (2003)
5. Fernandez, M., Morishima, A., Suciu, D.: Efficient Evaluation of XML Middle-ware Queries. In: SIGMOD (2001)
6. Gou, G., Chirkova, R.: Efficiently Querying Large XML Data Repositories: A Survey. IEEE TKDE 19(10) (2007)
7. Grust, T., Rittinger, J., Teubner, J.: Why Off-the-Shelf RDBMSs are Better at XPath Than You Might Expect. In: SIGMOD (2007)
8. Grust, T., Sakr, S., Teubner, J.: XQuery on SQL Hosts. In: VLDB (2004)
9. Krishnamurthy, R., Kaushik, R., Naughton, J.F.: Efficient XML-to-SQL Query Translation Literature: State of the Art and Open Problems. In: XSym (2003)
10. Leonardi, E., Bhowmick, S.S., Li, F.: Fast Evaluation of Multi-source Star Twig Queries in a Path Materialization-based xml Database. Technical Report (2010), http://www.cais.ntu.edu.sg/~assourav/TechReports/StarJoin-TR.pdf
11. Manolescu, I., Florescu, D., Kossmann, D.: Answering XML Queries over Heterogeneous Data Sources. In: VLDB (2001)
12. O'Neal, P., O'Neal, E., Pal, S., et al.: ORDPATHs: Insert-Friendly XML Node Labels. In: SIGMOD (2004)
13. Pal, S., Cseri, I., Seeliger, O., et al.: XQuery Implementation in a Relational Database System. In: VLDB (2005)
14. Seah, B.-S., Widjanarko, K.G., Bhowmick, S.S., Choi, B., Leonardi, E.: Efficient Support for Ordered XPath Processing in Tree-Unaware Commercial Relational Databases. In: Kotagiri, R., Radha Krishna, P., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 793–806. Springer, Heidelberg (2007)
15. Shanmugasundaram, J., Tufte, K., et al.: Relational Databases for Querying XML Documents: Limitations and Opportunities. In: VLDB (1999)
16. Shanmugasundaram, J., Kiernan, J., et al.: Querying XML Views of Relational Data. In: VLDB (2001)

Updating Typical XML Views

Jixue Liu[1], Chengfei Liu[2], Theo Haerder[3], and Jeffery Xu Yu[4]

[1] School of CIS, University of South Australia
jixue.liu@unisa.edu.au
[2] Faculty of ICT, Swinburne University of Technology
cliu@swin.edu.au
[3] Dept. of CS, Technical University of Kaiserslautern
haerder@informatik.uni-kl.de
[4] Dept. of Sys. Eng. and Eng. Management, Chinese University of HK
yu@se.cuhk.edu.hk

Abstract. View update is the problem of translating an update to a view to some updates to the source data of the view. In this paper, we formally define the problem, show the factors determining XML view update translation, and propose a translation solution for two specific but typical settings of the view update problem. We prove that the translated source updates are precise and they generalize the solutions to the problem with similar settings in the relational database.

Keywords: XML data, view update, update translation, virtual views.

1 Introduction

A (virtual) view is defined with a query over some source data of a database. The query is called the view definition which determines what data appears in the view. The data of the view, called a **view instance**, is often not stored in the database but is derived from the source data on the fly using the view definition every time when the view is selected.

In database applications, many users do not have privileges to access all the data of a database. They are often given a view of the database so that they can retrieve only the data in the view. In data integration applications, user's access to the source data becomes even more impractical because of security. When these users need to update the data of the database, they put their updates against the view, not against the source data, and expect that the view instance is changed when it is accessed next time. This type of updates is called a view update. *Because of its important use*, view update has a long research history [1,9,11,12,6,4,13]. The work in [5] discusses detailed semantics of view updates in many scenarios.

Unfortunately, view updates cannot be directly applied to the view instance as it is not stored physically and is derived on the fly when required (virtual view). Even in the cases where the view instance is stored (materialized view), which is not the main focus of this paper, applying updates to the instance may

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 126–140, 2012.
© Springer-Verlag Berlin Heidelberg 2012

cause inconsistencies between the source data and the instance. To apply a view update to a virtual view, a translation process is required to translate the view update to some *source updates*. When the source data is changed, the data in the view will be changed next time when the view is selected. To the user of the view, it seems that the view update has been successfully applied to the view instance.

Let V be a view definition, V^i the view instance, S^i the source data of the view, $V(S^i)$ the evaluation of V against S^i. Then $V^i = V(S^i)$. Assume that the user wants to apply a view update δV to V^i as $\delta V(V^i)$. View update translation is to find a process that takes V and δV as input and produces a source update δS to S^i such that next time when the user accesses the view, the view instance appears changed and is as expected by the user. That is, for any S^i and $V^i = V(S^i)$,

$$V(\delta S(S^i)) = \delta V(V^i) \tag{1}$$

Two typical anomalies, view side-effect and source document over-update, are easily introduced by the translation process although they are update policy dependent [9]. View side-effect [13] occurs if the translated source update causes more-than-necessary change to the source data which leads to more-than-expected change to the view instance. View side-effect makes Equation (1) violated.

Over-updates may also happen to a source document. An over-update to a source document causes the source data irrelevant to the view to be changed, but keeps the equation satisfied. A source document over-update is incorrect as it changes information that the user did not expect to change.

A **precise** translation of a view update should produce source updates that (1) result in necessary (as the user expects) change to the view instance, (2) do not cause view side-effect, and (3) do not cause over-updates to the source documents.

In the *relational database*, much work has been done on view update and the problem has been well understood [1,9,11]. In case of updating *XML views over relational databases*, updates to XML views need to be translated to updates to the base relational tables. The works in [4,13] propose two different approaches to the problem. The work in [4] translates an XML view to some relational views and an update to the XML view to updates to the relational views. It then uses the relational approach to derive updates to the base tables. The work in [13] derives a schema for the XML view and annotates the schema based on keys of relational tables and multiplicities. An algorithm is proposed to use the annotation to determine if a translation is possible and how the translation works. Both works assume keys, foreign keys and the join operator based on these two types of constraints. Another work, technical report [6], proposes brief work on updating hypertext views defined on relational databases. In the case of updating *pure XML views*, the views where the views and their source data are all modelled in XML, no direct work has been done. To the best of our knowledge, the only work relating to XML view update is [8] which proposes a middle language and a transformation system to derive view instance from source data, and to derive source data from a **materialized** view instance, and assumes XQuery as the view definition language. We argue that with the view update problem, only view updates are available but not the view instance (not materialized). Consequently view update techniques are still necessary. The work in [3] is on the exact topic as this paper, but it restricts its views only to node relabeling and selection; no restructuring is allowed in the view definitions.

In this paper, we look into the view update problem in pure XML context. This means that both source data and the view are in XML format. We assume that base XML documents have no schema and no constraints information available. Assuming no constraints makes the solutions developed more general.

The view update problem in the relational database is difficult as not all view updates are translatable. For example, if a view V is defined by a Cartesian product of two tables R and S, an update inserting a new tuple to the view instance is not translatable because there is no unique way to determine the change(s) to R and S. The view update problem in XML becomes much harder. The main reason is that the source data and view instances are modeled in trees and trees can nest in arbitrary levels. This fundamental difference makes the methods of translating view updates in the relational database not applicable to translating XML view updates. A typical example is that the selection and the projection operations in the relational database do not have proper counterparts in XML. The view update problem in XML has many distinct cases that do not exist in the view update problem in the relational database (see Sections 3 and 4 for details). To the best of our knowledge, our work is the first proposing a solution to the view update problem in XML.

We notice that the view update problem is different from the view maintenance problem. The former aims to translate a view update to a virtual view to a source update while the latter aims to translate a source update to a view update to a materialized view. The methods for one do not work for the other.

We make the following contributions in this paper. Based on the view definition and the update language presented later, we present a formal definition of the view update problem and identify the factors determining the view update problem. Secondly, we propose a translation solution to translate 'typical' view updates. We prove that the translated source update from the algorithm is precise in the typical settings. Furthermore, we show that the solution we propose generalizes the solution to the problem in the relational database in similar settings.

The paper is organized as follows. Section 2 shows the view definition language, the update language, and the preciseness of view update translation. In Section 3, we propose an algorithm and show that the translation obtained by the algorithm is a precise translation. In Section 4, we identify a 'join' case where a translated update is precise. Section 5 concludes the paper.

2 Preliminaries

In this section, we define basic notation, introduce the languages for view definitions and updates, and define the XML view update problem.

Definition 1 (tree)**.** An XML document can be represented as an ordered tree. Each node of the tree has a unique identifier v_i, an element name *ele* also called a **label**, and either a text string *txt* or a sequence of child trees $T_{j_1}, \cdots, T_{j_n}$. That is, a node is either $(v_i : ele : txt)$ or $(v_i : ele : T_{j_1}, \cdots, T_{j_n})$. When the context is clear, some or all of the node identifiers of a tree may not present explicitly in this paper. A tree without all node identifiers is called a **value tree**. Two trees T_1 and T_2 are (value) **equal**, denoted by $T_1 = T_2$, if they have identical value trees. If a tree T_1 is a subtree in T_2, T_1 is said **in** T_2 and denoted by $T_1 \in T_2$. □

For example, the document `<root><A><B>1</B></A><A><B>2</B></A></root>` is represented by $T = (v_r\!:\!root\!:\ (v_0\!:\!A\!:\!(v_1\!:\!B\!:\!1)), (v_2\!:\!A\!:\!(v_3\!:\!B\!:\!2)))$. The value tree of T is $(root\!:\ (A\!:\!(B\!:\!1)), (A\!:\!(B\!:\!2)))$.

Definition 2. A **path** p is a sequence of element names $e_1/e_2/\cdots/e_n$ where all names are distinct. The function $L(p)$ returns the last element name e_n.

Given a path p and a sequence of nodes $v_1, \cdots, v_n$ in a tree, if for every node $v_i \in [v_2, \cdots, v_n]$, v_i is labeled by e_i and is a child of v_{i-1}, then $v_1/\cdots/v_n$ is a **doc path** conforming to p and the tree rooted at v_n is denoted by $T^p_{v_n}$. □

2.1 View Definition Language

We assume that a view is defined in a dialect of the *for-where-return* clauses of XQuery [2].

Definition 3 (V)**.** A view is defined by

```
<v>{ for $x_1$ in $p_1$,   $\cdots$,   $x_n$ in $p_n$
      where   $cdn(x_1, \cdots, x_n)$
      return  $rtn(x_1, \cdots, x_n)$   }</v>
```

where $p_1, \cdots, p_n$ are paths (Definition 2) proceeded by $doc()$ or x_i;

$cdn(x_1, \cdots, x_n) ::= x_i/\mathcal{E}_i = x_j/\mathcal{E}_j$ and $\cdots$ and $x_k/\mathcal{E}_k = strVal$ and $\cdots$;

$rtn(x_1, \cdots, x_n) ::= <\mathfrak{e}>\ \{x_u/\gamma_u\} \cdots \{x_v/\gamma_v\}\ </\mathfrak{e}>$;

$\gamma, \mathcal{E}$ are paths, and the last elements of all $x_u/\gamma_u, \cdots, x_v/\gamma_v$ are distinct. □

We note that the paths in the *return* clause are denoted by x_i/γs because these expressions are specially important in view update translation. We purposely leave out the $ sign proceeding a variable in the XQuery language.

Definition 4 (context-based production)**.** By the formal semantics of XQuery [7], the semantics of the language is

```
for $x_1$ in $p_1$ return
    for $x_2$ in $p_2$ return
        ...
            for $x_n$ in $p_n$ return
                if cdn($x_1$,...,$x_n$)=true
                    return rtn($x_1$,...,$x_n$)
```

The for-statement produces tuples $<x_1, ..., x_n>$, denoted by $fortup(V)$, where the variable x_i represents a binding to one of the sub trees located by p_i within the context defined by $x_1, \cdots, x_{i-1}$. This process is called **context-based production**. □

For each tuple satisfying the condition $cdn(x_1, ..., x_n)$, the function $rtn(x_1, \cdots, x_n)$ produces a tree, called an $\mathfrak{e}$-tree, under the root node of the view. That is, V maps a tuple to an $\mathfrak{e}$-tree. The children of the $\mathfrak{e}$-tree are the **γ-trees** selected by all the expressions x_i/γ_is (for all i) from the tuple. A tuple is mapped to one and only one $\mathfrak{e}$-tree and an $\mathfrak{e}$-tree is for one and only one tuple. A γ-tree of a tuple is uniquely mapped to a child of the $\mathfrak{e}$-tree of the tuple and a child of an $\mathfrak{e}$-tree is for one and only one γ-tree of its tuple. This is illustrated in Figure 1(a) where x_c and x_t are two variables, T^{x_t/γ_t} and T^{x_c/γ_c} are γ-trees, and the γ-trees appear as children of the $\mathfrak{e}$ node. We note that the one-to-one mapping is between a tree in the tuple (not the source document) and a tree under an $\mathfrak{e}$ node in the view.

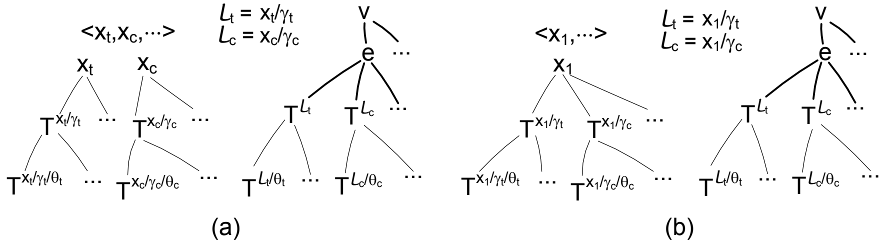


Fig. 1. Each of tuples is mapped to an $\mathfrak{e}$-tree

The path of a node s in the view has the following format:

$$v/\mathfrak{e}/\mathcal{L}_i/\theta_i \tag{2}$$

where

$$\mathcal{L}_i = L(x_i/\gamma_i) \tag{3}$$

returns the last element name $\mathcal{L}_i$ of x_i/γ_i, an expression in $rtn(x_1, ..., x_n)$, and θ_i is a path following $\mathcal{L}_i$ in the view. When $\mathcal{L}_i/\theta_i$ is not empty, the path in the source document corresponding to $v/\mathfrak{e}/\mathcal{L}_i/\theta_i$ is

$$x_i/\gamma_i/\theta_i \tag{4}$$

The view definition has some properties important to view update translation. Firstly because of context-based production, a binding of variable x_i may be copied into $x_i^{(1)}, \cdots, x_i^{(m)}$ to appear in multiple tuples:

$$\begin{array}{l} <\cdots, x_i^{(1)}, \cdots, x_{j[1]}, \cdots> \\ \cdots \\ <\cdots, x_i^{(m)}, \cdots, x_{j[m_j]}, \cdots> \end{array}$$

where $x_{j[1]}, \cdots, x_{j[m_j]}$ are different bindings of x_j. Each tuple satisfying the condition $cdn(x_1, \cdots, x_n)$ is used to build an $\mathfrak{e}$-tree. As a result of x_i being copied, the subtrees of x_i will be copied accordingly to appear in multiple $\mathfrak{e}$-trees in the view.

Secondly, a tree may have zero or many sub trees located by a given path p. That is, given a tree bound to x_i, the path expression x_i/p may locate zero or many sub trees $T_1^{x_i/p}, \cdots, T_{n_p}^{x_i/p}$ in x_i. This is true both in the source documents and in the view.

Thirdly, two path expressions x_i/γ_i and x_j/γ_j generally may have the same last element name, i.e., $L(x_i/\gamma_i) = L(x_j/\gamma_j)$. For example, if x_i represents an employee while x_j represents a department, then $x_i/name$ and $x_j/name$ will present two types of names in the same $\mathfrak{e}$-tree. This make the semantics of the view data not clear. This is the reason why we assume that all $L(x_i/\gamma_i)$s are distinct.

Example 1. Consider the view definition below and the source document shown in Figure 2(a). The view instance is shown in Figure 2(b).

```
<v>{for x in doc("r")/r/A,   y in x/C,   z in x/H
     where y/D=z and z="1"
     return <𝔢>{x/B}{x/C}{y/F/G}{z}</𝔢>
}</v>
```

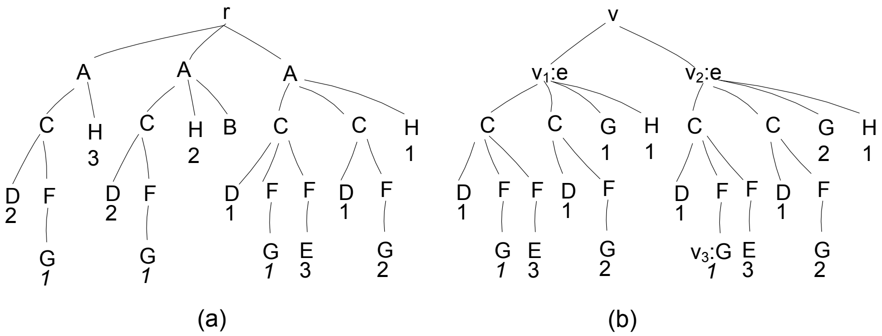


Fig. 2. Source document r and view v

From the view definition, $\gamma_1 = B$, $\gamma_2 = C$, $\gamma_3 = F/G$, and $\gamma_4 = \phi$. $L(x/\gamma_1) = \mathcal{L}_1 = B$, $L(x/\gamma_2) = \mathcal{L}_2 = C$, $L(y/\gamma_3) = \mathcal{L}_3 = G$, and $L(z/\gamma_4) = \mathcal{L}_4 = H$.

Formula (2) is exemplified as the following. The node v_3 in the view has the path $v/e/C/F/G$ where C is $\mathcal{L}_2 = L(x/\gamma_2)$ and F/G is θ. The node v_2 is an $\mathfrak{e}$ node and its path is v/e where $\mathcal{L}_i/\theta_i$ is ϕ.

The example shows the following.

- The expression x/B (=x/γ_1) of the *return* clause has no tree in the two $\mathfrak{e}$-trees.
- The path expression x/C (=x/γ_2) has multiple trees in each $\mathfrak{e}$-tree.

- The trees of x/C are duplicated in the view and so are their sub trees.
- Each of some x/C trees has more than one $x/C/F$ (=$x/\gamma_2/\theta$) sub trees.

2.2 The Update Language

The update language we use follows the proposal [10] extended from XQuery.

Definition 5 (δV)**.** A view update statement has the format of

```
for $\bar{x}_1$ in $\bar{p}_1$,  $\cdots$,  $\bar{x}_u$ in $\bar{p}_u$
where $\bar{x}_c/\bar{p}_c = strValu$
update $\bar{x}_t/\bar{p}_t$ ( delete $T$ |  insert $T$ )
```

where $\bar{x}_c, \bar{x}_t \in [\bar{x}_1, \cdots, \bar{x}_u]$, $\bar{p}_1, \cdots, \bar{p}_u$ are paths (Definition 2) proceeded by v or $\bar{x}_i$; $\bar{p}_c, \bar{p}_t$ are paths; all element names in the paths are elements names in the view. $\bar{x}_c/\bar{p}_c$ and $\bar{x}_t/\bar{p}_t$ are called the **(update) condition path** and **(update) target path** respectively. □

The next procedure maps a path in the update statement to a path in the source document.

Procedure 1 (mapping)**.**
(i). Replace the variables in $\bar{x}_c/\bar{p}_c$ and $\bar{x}_t/\bar{p}_t$ by their paths in the *for*-clause until the first element name becomes v. Thus the full paths of $\bar{x}_c/\bar{p}_c$ and $\bar{x}_t/\bar{p}_t$ in the view are built and will have the format of $v/\mathfrak{e}/\mathcal{L}_c/\theta_c$ and $v/\mathfrak{e}/\mathcal{L}_t/\theta_t$ as shown in Formula (2).
(ii). Search in the *return* clause of V using $\mathcal{L}_c$ and $\mathcal{L}_t$ to identify the expressions x_c/γ_c and x_t/γ_t. Append θ_t and θ_c to them respectively as $x_c/\gamma_c/\theta_c$ and $x_t/\gamma_t/\theta_t$. Thus, $x_c/\gamma_c/\theta_c$ and $x_t/\gamma_t/\theta_t$ are the source paths of $v/\mathfrak{e}/\mathcal{L}_c/\theta_c$ and $v/\mathfrak{e}/\mathcal{L}_t/\theta_t$. □

With this mapping, the update statement δV can be represented by the following abstract form:

$$(\bar{p}_s;\ \ v/\mathfrak{e}/\mathcal{L}_c/\theta_c = strValu;\ \ v/\mathfrak{e}/\mathcal{L}_t/\theta_t;\ \ del(T)|ins(T)) \tag{5}$$

where

- $v/\mathfrak{e}/\mathcal{L}_c/\theta_c$ is the **full** update condition path (int the view) for $\bar{x}_c/\bar{p}_c$, $v/\mathfrak{e}/\mathcal{L}_t/\theta_t$ the **full** target path for $\bar{x}_t/\bar{p}_t$;
- $\bar{p}_s$ is the maximal common front part of $v/\mathfrak{e}/\mathcal{L}_c/\theta_c$ and $v/\mathfrak{e}/\mathcal{L}_t/\theta_t$.

The semantics of an update statement is that under a context node identified by $\bar{p}_s$, if a sub tree identified by $v/\mathfrak{e}/\mathcal{L}_c/\theta_c$ satisfies the update condition, all the sub trees identified by $v/\mathfrak{e}/\mathcal{L}_t/\theta_t$ will be applied the update action (del(T) or ins(T)). The sub tree $T^{v/\mathfrak{e}/\mathcal{L}_c/\theta_c}$ is called the **condition tree** of $T^{v/\mathfrak{e}/\mathcal{L}_t/\theta_t}$. A sub tree is updated only if it has a condition tree and the condition tree satisfies the update condition. An update target and its condition trees are always within a tuple when the view definition is evaluated and are in an $\mathfrak{e}$-tree in the view after the evaluation.

We note that because of the context-based production in the view definition, the same update action may be applied to a target node for multiple times. For example, if x is a binding and the context-based production produces two tuples as $< x^{(1)}, \cdots >$ and $< x^{(2)}, \cdots >$. If the update condition and target are all in x, x will be updated twice with the same action, each action being fired for each tuple. We assume that only the effect of the first application is taken and the effect of all other applications is ignored.

Based on the structure of the target path $tp = v/\mathfrak{e}/\mathcal{L}_t/\theta_t$, updates may happen to different types of nodes in the view.

- When $\mathcal{L}_t/\theta_t \neq \phi$, the update happens to the nodes within a γ-tree.
- When $tp = v/\mathfrak{e}$, the update will add or delete a γ-tree.
- When $tp = v$ (in this case, $\bar{p}_s = v$), the update will add or delete an $\mathfrak{e}$-tree.

In this paper, we only deal with the first case and leave the solutions to the last two cases to be future work.

2.3 The View Update Problem

Definition 6 (Precise Translation)**.** Let V be a view definition and S be the source of V. Let δV be an update statement to V. Let δS be the update statement to S translated from δV. δS is a precise translation of δV if, for any instance S^i of S and $V^i = V(S^i)$,

(1) δS is correct. That is, $V(\delta S(S^i)) == \delta V(V^i)$ is true; and

(2) δS is minimal. That is, there does not exist another translation $\delta S'$ such that ($\delta S'$ is correct, i.e., $V(\delta S'(S^i)) = V(\delta S(S^i)) = \delta V(V^i)$ and there exists a tree T in S^i and T is updated by δS but not $\delta S'$). □

We note that Condition (1) also means that the update δS will not cause view-side-effect. Otherwise, $V(\delta S(S^i))$ would contain more, less, or different updated trees than those in $\delta V(V^i)$.

Definition 7 (the view update problem)**.** Given a view V and a view update δV, the problem of view update is to (1) develop a translation process P, and show that the source update δS obtained from P is precise, or (2) prove that a precise translation of δV does not exist. □

3 Update Translation When $\mathcal{L}_t/\theta_t \neq \phi$ and $x_c = x_t$

In this section, we investigate update translation when the update is to change a γ-tree of the view and the mappings of the update condition path and the update target path refer to the same variable in the view definition. We present Algorithm 1 and the statement δS as the solution for view update translation in this case.

By the algorithm, the following source update is derived.

```
$\delta S$:  for $x_1$ in $p_1$,     $\cdots$,     $x_n$ in $p_n$
      where $cdn(x_1, \cdots, x_n)$     and     $x_c/\gamma_c/\theta_c = strValu$
      update $x_t/\gamma_t/\theta_t$  (insert $T$ | delete $T$)
```

Algorithm 1: A translation algorithm

```
Input: view definition $V$, view update $\delta V$
Output: translated source update $\delta S$
begin
    make a copy of $V$ and reference the copy by $\delta S$ ;
    remove $rtn()$ from $\delta S$ ;
    from the view update $\delta V$, following Procedure 1, find mappings $x_c/\gamma_c/\gamma_c$
    and $x_t/\gamma_t/\gamma_t$ for the condition path $\bar{x}_c/\bar{p}_c$ and the target path $\bar{x}_t/\bar{p}_t$ ;
    make a copy of $\delta V$ and reference the copy by $\delta V_c$ ;
    in $\delta V_c$, replace $\bar{x}_c/\bar{p}_c$ and $\bar{x}_t/\bar{p}_t$ by $x_c/\gamma_c/\gamma_c$ and $x_t/\gamma_t/\gamma_t$ respectively ;
    append the condition in the where clause of $\delta V_c$ to the end of the where
    clause in $\delta S$ using logic and ;
    append the update clause of $\delta V_c$ after the where clause of $\delta S$
```

We now develop the preciseness of the translation. We recall the notation that $fortup(V)$ means the tuples of the context-based production (Definition 4) of V. The symbols x_c, $x_{c[1]}$ and $x_{c[2]}$ are three separate bindings of x_c. The symbols $x_c^{(1)}$ and $x_c^{(2)}$ are two copies of x_c.

Lemma 1. *Given a tuple $t = <x_t, x_c, \cdots> \in fortup(V)$ and its $\mathfrak{e}$-tree e, (1) if $T^{x_t/\gamma_t/\theta_t}$ (a tree for the path $x_t/\gamma_t/\theta_t$) in t is updated by δS, then all the trees identified by $x_t/\gamma_t/\theta_t$ in t are updated by δS, and all the trees identified by $\mathcal{L}_t/\theta_t$ in e are updated by δV. (2) if $T^{\mathcal{L}_t/\theta_t}$ (a tree for the path $\mathcal{L}_t/\theta_t$) in e is updated by δV, then all the trees identified by $\mathcal{L}_t/\theta_t$ in e are updated by δV, and all the trees identified by $x_t/\gamma_t/\theta_t$ in t are updated by δS.*

The lemma is correct because of the one-to-one correspondence between a tuple and an $\mathfrak{e}$-tree and between t's γ-trees and e's children, and because all the trees identified by $x_t/\gamma_t/\theta_t$ in t share the same condition tree(s) identified by $x_c/\gamma_c/\theta_c$ in x_c of t, and all the trees identified by $\mathcal{L}_t/\theta_t$ in e share the same condition tree(s) identified by $\mathcal{L}_c/\theta_c$ in e.

Lemma 2. *Assume a tuple $t = <x_t, x_c, \cdots> \in fortup(V)$. After a tree $T^{x_t/\gamma_t/\theta_t}$ in x_t is updated by δS, t becomes $t' = <x_t', x_c, \cdots>$. If $x_t/\gamma_t/\theta_t$ is not a prefix of any of the paths in the where clause of δS and if t satisfies $cdn()$ of V, t' also satisfies $cdn()$ of V.*

The lemma is correct because the subtrees in the tuple used to test $cdn()$ are not changed by δS when the condition of the lemma is met.

Lemma 3. *Assume a tuple $t = <x_t, x_c, \cdots> \in fortup(V)$ and its $\mathfrak{e}$-tree e. If $T^{x_c/\gamma_c/\theta_c}$ in t satisfies $x_c/\gamma_c/\theta_c = strValu$, $T^{\mathcal{L}_c/\theta_c}$ in e satisfies $\mathcal{L}_c/\theta_c = strValu$ and vice versa.*

The correctness of the lemma is guaranteed by the one-to-one correspondence between t's γ-trees and e's children.

Lemma 4. *Assume a tuple $t = <x_t, x_c, \cdots> \in fortup(V)$, its $\mathfrak{e}$-tree e, $T^{x_t/\gamma_t/\theta_t}$ in x_t, and $T'^{\mathcal{L}_t/\theta_t}$ in e. Obviously $T = T'$. As δS and δV have the same update action, if x_c satisfies the update condition, $\delta S(T) = \delta V(T')$.*

Theorem 1. *Given view V and the view update δV where $\mathcal{L}_t/\theta_t \neq \phi$ and $x_c = x_t$, the update δS is a precise translation of the view update δV if and only if $x_t/\gamma_t/\theta_t$ does not proceed any path in the where clause of δS.*

Proof. We show only the 'if' proof. The proof of 'only if' can be done in a similar way. We follow Definition 6 to show that if the condition is true, the translation is precise. Without losing generality, we assume that $x_t = x_c = x_1$. Figure 1(b) illustrates the relationship between a variable binding x_1 in the tuple $< x_1, \cdots >$ and the $\mathfrak{e}$-tree built from the tuple. $T^{x_1/\gamma_t/\theta_t}$ and $T^{x_1/\gamma_c/\theta_c}$are an update target tree and a condition tree respectively. $T^{x_1/\gamma_t/\theta_t}$'s children will be deleted or a new child will be inserted.

(1) Correctness: $V(\delta S(S^i)) = \delta V(V(S^i))$

We firstly show that duplicated γ-trees are updated consistently in the view and then show that each side of the equation is contained in the other side. The reason why consistency is important here is that if the update changes one copy of a duplicated γ-tree without updating the others, the translation will have side effect.

Consistency: Consider two tuples $t_1 =< x_1^{(1)}, \cdots >$ and $t_2 =< x_1^{(2)}, \cdots >$ in $fortup(\delta S)$ where $x_1^{(1)}$ and $x_1^{(2)}$ are copies of x_1. Let e_1 and e_2 be two $\mathfrak{e}$-trees constructed from t_1 and t_2 respectively by V. Then, because of $x_t = x_c = x_1$, either both e_1 and e_2 are updated by δV or none is updated.

$\supseteq$: Let $T^{\mathcal{L}_t/\theta_t}$ be a tree in an $\mathfrak{e}$-tree e of $V(S^i)$ updated to $\bar{T}^{\mathcal{L}_t/\theta_t}$ by δV (e becomes e' after the update). We show that $\bar{T}^{\mathcal{L}_t/\theta_t}$ is in e' of $V(\delta S(S^i))$. In fact, that $T^{\mathcal{L}_t/\theta_t}$ is in $V(S^i)$ means that there exists one and only one tuple $t = <x_1, \cdots>$ in $fortup(V)$ satisfying $cdn()$, that in the tuple, $x_1/\gamma_t/\theta$ identifies the source tree $T^{x_1/\gamma_t/\theta_t}$ of $T^{\mathcal{L}_t/\theta_t}$. $T^{\mathcal{L}_t/\theta_t}$ being updated by δV means that there exists a condition tree $T^{\mathcal{L}_c/\theta_c}$ in e and the condition tree satisfies $v/\mathfrak{e}/\mathcal{L}_c/\theta_c = strValu$.

On the other side, because V and δS have the same *for* clause, t is in $fortup(\delta S)$. Because $T^{\mathcal{L}_c/\theta_c}$ makes $v/\mathfrak{e}/\mathcal{L}_c/\theta_c = strValu$ true, so $T^{x_1/\gamma_c/\theta_c}$ makes $x_1/\gamma_c/\theta_c = strVal$ true (Lemma 3). This means $T^{x_1/\gamma_t/\theta_t}$ is updated by δS and becomes $\hat{T}^{x_1/\gamma_t/\theta_t}$. Thus t becomes $t' =< \bar{x}_1, \cdots >$. Because of Lemma 4, $\bar{T}^{x_1/\gamma_t/\theta_t}$=$\hat{T}^{x_1/\gamma_t/\theta_t}$. Because of the condition of the theorem and Lemma 2, t' satisfies $cdn()$ and generalizes e' in the view. So $\bar{T}^{\mathcal{L}_t/\theta_t}$ is in $V(\delta S(S^i))$.

$\subseteq$: Let $T_1^{\mathcal{L}_t/\theta_t}$ and $T_2^{\mathcal{L}_t/\theta_t}$ be two trees in $V(\delta S(S^i))$ and their source tree(s) are updated by δS. We show that $T_1^{\mathcal{L}_t/\theta_t}$ and $T_2^{\mathcal{L}_t/\theta_t}$ are in $\delta V(V(S^i))$. There are three cases: (a) $T_1^{\mathcal{L}_t/\theta_t}$ and $T_2^{\mathcal{L}_t/\theta_t}$ share the same source tree $T^{x_1/\gamma_t/\theta_t}$ (they must appear in different $\mathfrak{e}$-trees in the view), and (b) $T_1^{\mathcal{L}_t/\theta_t}$ and $T_2^{\mathcal{L}_t/\theta_t}$ have different source trees $T_1^{x_1/\gamma_t/\theta_t}$ and $T_2^{x_1/\gamma_t/\theta_t}$. Case (b) has two sub cases: (b.1) $T_1^{\mathcal{L}_t/\theta_t}$ and $T_2^{\mathcal{L}_t/\theta_t}$ appear in the same $\mathfrak{e}$-tree in the view, and (b.2) $T_1^{\mathcal{L}_t/\theta_t}$ and $T_2^{\mathcal{L}_t/\theta_t}$ appear in different $\mathfrak{e}$-trees.

Case (a): That $T^{x_1/\gamma_t/\theta_t}$ is updated by δS means that there exist two tuples $<x_1^{(1)}, \cdots>$ and $<x_1^{(2)}, \cdots>$ in $fortup(\delta S)$ such that $x_1^{(1)} = x_1^{(2)}$, both tuples satisfy *cdn*(), and there exists condition tree $T^{x_1/\gamma_c/\theta_c}$ in each tuple satisfying $x_c/\gamma_c/\theta_c = strValu$, $T^{x_1/\gamma_t/\theta_t}$ is updated to $\bar{T}^{x_1/\gamma_t/\theta_t}$ by δS (two update attempts with the same action for the two tuples, only the effect of the first attempt is taken). After the update, the tuples become $t_1' = <\bar{x}_1^{(1)}, \cdots>$ and $t_2' = <\bar{x}_1^{(2)}, \cdots>$. By Lemma 2, t_1' and t_2' satisfy *cdn* of V and produce $e_1, e_2 \in V(\delta S(S^i))$ and $\bar{T}_1^{\mathcal{L}_t/\theta_t} \in e_1$ and $\bar{T}_2^{\mathcal{L}_t/\theta_t} \in e_2$.

On the other side, when V is evaluated against S^i, x_1 is copied to two tuples $t_1 = <x_1^{(1)}, \cdots>$ and $t_2 = <x_1^{(2)}, \cdots>$ in $fortup(V)$ and each of the tuples satisfies *cdn*(). They produce $\mathfrak{e}$-trees e_1' and e_2'. Because each tuple has a condition tree $T^{x_1/\gamma_c/\theta_c}$ satisfying $x_c/\gamma_c/\theta_c = strValu$, by Lemma 3, each of e_1' and e_2' has $T^{\mathcal{L}_c/\theta_c}$ satisfying $\mathcal{L}_c/\theta_c = strValu$ and each has a $T^{\mathcal{L}_t/\theta_t}$. Thus $T_1^{\mathcal{L}_t/\theta_t} \in e_1'$ and $T_2^{\mathcal{L}_t/\theta_t} \in e_2'$ will be updated to $\bar{T}_1^{\mathcal{L}_t/\theta_t}$ and $\bar{T}_2^{\mathcal{L}_t/\theta_t}$ by δV. e_1' and e_2' become e_1 and e_2 in $\delta V(V(S^i))$.

Case (b.1): That $T_1^{x_1/\gamma_t/\theta_t}$ and $T_2^{x_1/\gamma_t/\theta_t}$ are updated by δS and that they appear in different $\mathfrak{e}$-trees mean that there are two tuples $<x_{1[1]}, \cdots>$ and $<x_{1[2]}, \cdots>$ where $x_{1[1]}$ and $x_{1[2]}$ are different bindings of x_1, $T_1^{x_1/\gamma_t/\theta_t} \in x_{1[1]}$, $T_2^{x_1/\gamma_t/\theta_t} \in x_{c[2]}$, and each of tuples satisfies *cdn*() and $x_c/\gamma_c/\theta_c = strValu$. $T_1^{x_1/\gamma_t/\theta_t}$ and $T_2^{x_1/\gamma_t/\theta_t}$ become $\bar{T}_1^{x_1/\gamma_t/\theta_t}$ and $\bar{T}_2^{x_1/\gamma_t/\theta_t}$ after the update and mapped to $\bar{T}_1^{\mathcal{L}_t/\theta_t}$ and $\bar{T}_2^{\mathcal{L}_t/\theta_t}$ in two different $\mathfrak{e}$-trees of $V(\delta S(S^i))$. Following the same argument of Case (a), $\bar{T}_1^{\mathcal{L}_t/\theta_t}$ and $\bar{T}_2^{\mathcal{L}_t/\theta_t}$ are in $\delta V(V(S^i))$.

Case (b.2): That $T_1^{x_1/\gamma_t/\theta_t}$ and $T_2^{x_1/\gamma_t/\theta_t}$ are updated by δS and that they appear in a single $\mathfrak{e}$-tree mean that there is one and only one tuple $<x_1, \cdots>$ where $T_1^{x_1/\gamma_t/\theta_t}, T_2^{x_1/\gamma_t/\theta_t} \in x_1$. The tuple satisfies *cdn*() and there is a tree $T^{x_1/\gamma_c/\theta_c}$ in the tuple satisfying $x_1/\gamma_c/\theta_c = strValu$. $T_1^{x_1/\gamma_t/\theta_t}$ and $T_2^{x_1/\gamma_t/\theta_t}$ become $\bar{T}_1^{x_1/\gamma_t/\theta_t}$ and $\bar{T}_2^{x_1/\gamma_t/\theta_t}$ after the update and mapped to $\bar{T}_1^{\mathcal{L}_t/\theta_t}$ and $\bar{T}_2^{\mathcal{L}_t/\theta_t}$ in a single $\mathfrak{e}$-tree of $V(\delta S(S^i))$. On the other side, as $T_1^{x_1/\gamma_t/\theta_t}$ and $T_2^{x_1/\gamma_t/\theta_t}$ are mapped to a single $\mathfrak{e}$-tree e and share the same condition tree $T^{x_1/\gamma_c/\theta_c}$, $T_1^{\mathcal{L}_t/\theta_t}$ and $T_2^{\mathcal{L}_t/\theta_t}$ share the same condition tree $T^{\mathcal{L}_c/\theta_c}$ in e and will be updated by δV. So $\bar{T}_1^{\mathcal{L}_t/\theta_t}$ and $\bar{T}_2^{\mathcal{L}_t/\theta_t}$ are in the $\mathfrak{e}$-tree of $\delta V(V(S^i))$.

(2) δS is minimal

We prove by contrapositive. Let $T^{\mathcal{L}_t/\theta_t}$ be a tree in the view updated by δV. Then from above proofs, $T^{x_1/\gamma_t/\theta_t}$ is updated by δS and there exists a tuple $<x_1, \cdots>$ such that $T^{x_1/\gamma_t/\theta_t}$ is in x_1 and x_1 has a condition tree $T^{x_1/\gamma_c/\theta_c}$ satisfying "*cdn*() and $x_1/\gamma_c/\theta_c = strValu$".

If $T^{x_1/\gamma_t/\theta_t}$ is not updated by $\delta S'$, either (a) x_1 is not a variable in the *for*-clause of $\delta S'$, i.e., x_1 is not in any tuple and neither is $T^{x_1/\gamma_t/\theta_t}$, or (b) x_1 is in the tuple $<x_1, \cdots>$ but $T^{x_1/\gamma_t/\theta_t}$ is not in x_1, or (c) x_1 is in the tuple $<x_1, \cdots>$ and $T^{x_1/\gamma_t/\theta_t}$ is in x_1 but one of "*cdn*()" and "$x_c/\gamma_c/\theta_c = strValu$" is not in $\delta S'$.

In Case (a), because x_1 is not a variable in $\delta S'$, so $T^{x_1/\gamma_t/\theta_t}$ will not be updated by $\delta S'$ (this does not prevent $T^{x_1/\gamma_t/\theta_t}$ from appearing in the view). This means that the $T^{\mathcal{L}_t/\theta_t}$ in $V(\delta S'(S^i))$ is different from the $T^{\mathcal{L}_t/\theta_t}$ in $\delta V(V(S^i))$ because the assumption assumes that the $T^{\mathcal{L}_t/\theta_t}$ in $\delta V(V(S^i))$ is updated. This contradicts the correctness of $\delta S'$.

In Case (b), because $T^{x_1/\gamma_t/\theta_t}$ is not in x_1, so $T^{x_1/\gamma_t/\theta_t}$ is not in $V(S^i)$. This contradicts the assumption that $T^{\mathcal{L}_t/\theta_t}$ is in the view.

In Case (c), if *cdn*() is violated, the tuple of $T^{x_1/\gamma_t/\theta_t}$ will not be selected by V, so $T^{x_1/\gamma_t/\theta_t}$ is not in $V(S^i)$ which contradicts the assumption. If $x_1/\gamma_c/\theta_c = strValu$ is violated, $T^{x_1/\gamma_t/\theta_t}$ will not be updated by δV. This contradicts the assumption that $T^{\mathcal{L}_t/\theta_t}$ is updated by δV.

This concludes that δS is a precise translation. □

The View Update Problem Here Generalizes the View Update Problem of the Relational Database in a Similar Setting. Suppose that there are three relations *Student*(*sid*, *name*, *tel*), *Course*(*cid*, *name*, *credit*), and *Enrolment*(*sid*, *cid*, *year*, *mark*), a view defined by $V_r = Student \bowtie Enrolment \bowtie Course$, and an update statement `update Vr set tel="22345" where name="John"`. The update condition attribute *name* and the update target attribute *tel* are from the same 'variable' *Student*, and the update does not change the values of the join condition attributes *sid* and *cid*. Consequently the theorem applies. It says that this update is translatable and the translated source update is `update Student set tel="22345" where name in (select name from Student join Enrolment join Course) and name="John"`. This source update is obviously precise because, if two students having the same name 'John' and different telephone numbers '21344' and '21345', and if the telephone numbers are all changed to '22345' in the view, they are also all changed in the source relation. Next time when the view is derived, the telephone numbers will appear changed and the same.

4 Update Translation When $\mathcal{L}_t/\theta_t \neq \phi$ and $x_c \neq x_t$

We look into the translation problem when the mappings of the update condition and the update target are from different variables. The results of this section generalize the view update problem in the relational views when they are defined with the join operator.

In general, **view updates are not translatable in the case of** $x_c \neq x_t$. Consider two tuples where the binding x_t is copied to $x_t^{(1)}$ and $x_t^{(2)}$ to combine with two bindings $x_{c[1]}$ and $x_{c[2]}$ of x_c by the context-based production as

$$< \cdots, x_t^{(1)}, \cdots, x_{c[1]}, \cdots >$$
$$< \cdots, x_t^{(2)}, \cdots, x_{c[2]}, \cdots >$$

Assume that in the view, the update condition $x_c/\gamma_c/\theta_c$ is satisfied in $x_{c[1]}$ but violated in $x_{c[2]}$. Then, the copy of x_t for the first tuple will be updated but the one for the second will not. In the source, if x_t is updated, not only the first copy of x_t changes, but also the second copy. In other words, the translated source update has view side-effect. However, if x_t in the source is not updated, all its copies in the view will not be changed.

Although generally view updates, when $x_c \neq x_t$, are not translatable, for the following view and the view update, a precise translation exists.

```
V:    <v>{ for $x_1$ in $p_1$,   $\cdots$,    $x_n$ in $p_n$
           where $\cdots$ and $x_c/\gamma_c/\theta_c = x_{c+1}/\gamma_{c+1}/\theta_{c+1}$ and $\cdots$
           return $rtn(x_1, \cdots, x_n)$   }</v>
```

where x_c/γ_c is in $rtn(x_1, \cdots, x_n)$.

δV: $(\bar{p_s},\ \ v/\mathfrak{e}/\mathcal{L}_c/\theta_c = strValu,\ \ v/\mathfrak{e}/\mathcal{L}_t/\theta_t,\ \ del(T)|ins(T))$
where x_t is either x_c or x_{c+1}.

The conditions of the setting require that the condition path $x_c/\gamma_c/\theta_c$ must be a join path in the view definition and the γ-expression x_c/γ_c must be a prefix of this join path. At the same time, the variable of the target path must be x_c or x_{c+1}, the variable of the path joined to the update condition path.

Theorem 2. *Given view V and view update δV defined above where $\mathcal{L}_t/\theta_t \neq \phi$, update δS (in Section 3) is a precise translation of the view update δV if and only if $x_c/\gamma_t/\theta_t$ does not proceed any path in the where clause of δS.*

Proof. The notation of this proof follows that of the proof for Theorem 1 and Figure 1(a). Consider two tuples $t_1 = <x_t^{(1)}, x_{c[1]}, \cdots>$ and $t_2 = <x_t^{(2)}, x_{c[2]}, \cdots>$ in $fortup(\delta S(S))$ where $x_t^{(1)}$ and $x_t^{(2)}$ are copies of x_t and $x_{c[1]}$ and $x_{c[2]}$ can be the same. If one is updated by δS, the other is updated too. The reason is that for $T_1^{x_t/\gamma_t/\theta_t} \in x_t^{(1)}$ and $T_2^{x_t/\gamma_t/\theta_t} \in x_t^{(2)}$, because of the join condition in the view definition V: $x_c/\gamma_c/\theta_c = x_{c+1}/\gamma_{c+1}/\theta_{c+1}$ and because $x_{c+1} = x_t$ and $x_t^{(1)} = x_t^{(2)}$, a condition tree $T_1^{x_c/\gamma_c/\theta_c}$ exists for $T_1^{x_t/\gamma_t/\theta_t}$ and $T_2^{x_c/\gamma_c/\theta_c}$ exists for $T_2^{x_t/\gamma_t/\theta_t}$ and $T_1^{x_c/\gamma_c/\theta_c} = T_2^{x_c/\gamma_c/\theta_c}$. Consequently if $T_1^{x_c/\gamma_c/\theta_c}$ satisfies the update condition, so does $T_2^{x_c/\gamma_c/\theta_c}$. So either both $T_1^{x_t/\gamma_t/\theta_t}$ and $T_2^{x_t/\gamma_t/\theta_t}$ are updated or none is updated. Following Lemma 4, if e_1 and e_2 are mapped from $T_1^{x_t/\gamma_t/\theta_t}$ and $T_2^{x_t/\gamma_t/\theta_t}$ respectively, if one is updated, the other is updated too.

The remaining proof can be completed by following the argument of the proof of Theorem 1. □

Example 2. Consider the view definition in Example 1 and the view instance in Figure 2(b). If the view update is

```
δV:  for $s$ in $v/e$
      where $s/H = 1$
      update $s/C$ (insert ($K$  5))
```

By the theorem, this update is translatable because the full update condition path is $v/e/H$ and its source mapping, z, is a join path. The update target path is $/v/e/C$ and its source path, y, is the other end of the join condition $y/D = z$ in the view. Thus, the translated source update is below.

```
δS:  for x in doc("r")/r/A, y in x/C, z in x/H
      where y/D=z and z="1"
      update y (insert (K 5))
```

It is easy to check that the translated update works correctly. That is, $V(\delta S(S) = \delta V(V(S))$.

The View Update Problem Here Also Generalizes the Following View Update Problem of the Relational Database. Suppose that there are three relations $Student(sid, name, tel)$, $Course(cid, name, credit)$, and $Enrolment(stud, crs, year, semester, mark)$, a view defined by $Vr = Student \bowtie_{sid=stud} Enrolment \bowtie_{crs=cid} Course$, and an update statement `update Vr set mark="90" where sid="s01"`. The update condition attribute *sid* and the target attribute *mark* are from different 'variables' *Student* and *Enrolment* and they are bound to equal by the join condition, and the update does not change the values of the join condition attributes *sid*, *stud*, *cid* and *crs*. Consequently the theorem says that this update is translatable and the translated source update is `update Enrolment set mark="90" where stud in (select stud from Student join Enrolment on sid=stud join Course on crs=cid) and sid="s01"`. This source update is precise as if a student has two courses with the marks of '60' and '70', if they are changed to '90' in the view, they are also all changed to '90' in the source relation. Next time when the view is derived, the marks of the student will be '90' in the view.

5 Conclusion

In this paper, we invested two cases of the view update problem when the update target path is longer than the roots of 𝔢-trees. In the first case where the update target and the update condition are from the same variable, a solution is proposed and the translation obtained from the algorithm is proved to be precise. In the second case where the update target and the update condition are from the different variables, although in general view updates are not translatable, we discovered a specific type of view and a specific type of updates and derived a precise translation for the case. There are a few more cases that have not been investigated in the paper. These will be our future work.

References

1. Bancilhon, F., Spyratos, N.: Update semantics of relational views. TODS 6(4), 557–575 (1981)
2. Boag, S., Chamberlin, D., Fernndez, M.F., Florescu, D., Robie, J., Simon, J.: Xquery 1.0: An xml query language (2007), http://www.w3.org/TR/xquery/

3. Boneva, I., Groz, B., Tison, S., Caron, A.-C., Roos, Y., Stawork, S.: View update translation for xml. In: ICDT, pp. 42–53 (2011)
4. Braganholo, V.P., Davidson, S.B., Heuser, C.A.: From xml view updates to relational view updates: old solutions to a new problem. In: VLDB Conference, pp. 276–287 (2004)
5. Cong, G.: Query and Update through XML Views. In: Bhalla, S. (ed.) DNIS 2007. LNCS, vol. 4777, pp. 81–95. Springer, Heidelberg (2007)
6. Falquet, G., Nerima, L., Park, S.: Hypertext view update problem. Technical Report, University of Geneva (2000), www.cui.unige.ch/isi/reports/hvu.ps
7. Fankhauser, P.: Xquery formal semantics state and challenges. SIGMOD Record 30(3), 14–19 (2001)
8. Liu, D., Hu, Z., Takeichi, M.: Bidirectional interpretation of xquery. In: PEPM, pp. 21–30 (2007)
9. Llasunaga, Y.: A relational database view update translation mechanism. In: VLDB Conference, pp. 309–320 (1984)
10. Tatarinov, I., Ives, Z.G., Halevy, A.Y., Weld, D.S.: Updating xml. In: SIGMOD Conference, pp. 413–424 (2001)
11. Tomasic, A.: View Update Translation Via Deduction and Annotation. In: Gyssens, M., Van Gucht, D., Paredaens, J. (eds.) ICDT 1988. LNCS, vol. 326, pp. 338–352. Springer, Heidelberg (1988)
12. Tomasic, A.: Determining correct view update translations via query containment. In: Workshop on Deductive Databases and Logic Programming, pp. 75–83 (1994)
13. Wang, L., Rundensteiner, E.A., Mani, M.: Updating xml views published over relational databases: towards the existence of a correct update mapping. DKE 58, 263–298 (2006)

Partitioned Indexes for Entity Search over RDF Knowledge Bases

Fang Du[1], Yueguo Chen[2], and Xiaoyong Du[1,2]

[1] School of Information, Renmin University of China, Beijing, China
[2] Key Laboratory of Data Engineering and Knowledge Engineering
(Renmin University of China), MOE, China
{dfang,chenyueguo,duyong}@ruc.edu.cn

Abstract. The rapid growth of RDF data in RDF knowledge bases calls for efficient query processing techniques. This paper focuses on the star-style SPARQL join queries, which is very common when users want to search information of entities from RDF knowledge bases. We observe that the computational cost of such queries mainly comes from loading a large portion of predicate-ahead indexes. We therefore propose to partition the whole RDF knowledge bases based on the schema of individual entities, so that only entities of similar schemas are allocated into the same cluster. Such a partitioning strategy generates a pruning mechanism that effectively isolate the correlations of partitions and the queries. Consequently, queries are only conducted over a small number of partitions with small predicate-ahead indexes. Experiments over a large real-life RDF data set show the significant performance improvements achieved by our partitioned indexing techniques.

Keywords: Entity search, SPARQL query, index, clustering.

1 Introduction

The advances of techniques in information extraction, semantic web, and data integration allow the extraction and integration of massive simple facts from the web, represented as RDF triples in the form of $< s, \ p, \ o >$ that stand for a subject, a predicate and a value on that predicate respectively. A huge number of such RDF triples form a knowledge base (KB), whose size continuously and rapidly grows with the extraction and integration of new facts. Examples of such KBs are Freebase [2], Yago [6], and Linked Data [3,8]. They typically contain billions of RDF triples. According to the statistics of the W3C SWEO (Semantic Web Education and Outreach) group, the RDF triples scattered over the Web have reached up to 25 billion by September 2010 [3], and the number keeps growing. The large volume of RDF data in RDF KBs bring challenges for efficient query processing of RDF data.

The SPARQL query language [5], proposed by W3C, is a standard query interface for RDF data. Many studies [13,14,15,17,20] have been tried to address the challenge of efficient SPARQL query processing over large RDF KBs. Among them, RDF-3X [17] is widely accepted as the state of art for efficient SPARQL query processing. It achieves good efficiency by applying query optimization techniques over the indexes

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 141–155, 2012.
© Springer-Verlag Berlin Heidelberg 2012

of different combinations of S (subjects), P (predicates), O (objects) for RDF triples. The efficiency of RDF-3X is still not good enough when queries require to scan a large portion of an index. Figure 1 shows an example of such queries, which often happen when a basic graph pattern (BGP) is not selective (i.e., a large number of RDF triples satisfy the BGP, which typically happens for popular predicates such as $name$ and $born\_in\_place$ in the given example). We observe that for many star-style SPARQL

```
SELECT ?n ?r
WHERE
{?x name ?n
?x research_area Physics
?x born_in_place ?r
?x awarded Nobel_prize}
```

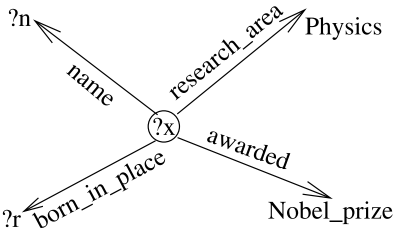


Fig. 1. A query example that requires scanning large predicate indexes for the predicates $name$ and $born\_in\_place$

[12] queries (i.e., join over one subject variable. One example is shown in Figure 1), although some BGPs may not be selective, the combination of the predicates specified in the query (for those BPGs whose predicates are not variables) is very useful for effectively prune indexes that may not be relevant to the given query. This is because the combination of predicates somehow implies the types of desired subjects. Considering that many entities (or subjects exchangeably) extracted from the same sources (e.g., IMDB) typically have similar set of predicates, we therefore are able to partition RDF triples based on the predicate set of their entities, so that entities in the same partition have similar sets of predicates. In this way, we are able to efficiently prune clusters of RDF triples based on the schema (predicate set) shared by the entities within the same cluster. Based on the partition of RDF KBs and the efficient pruning mechanism, we can significantly improve the performance of star-style SPARQL queries (we call them entity search in this paper) over the state of art. The contribution of the paper can be summarized as follows:

- We propose techniques on efficiently and effectively partition the huge RDF KBs based on the schema of entities, to support efficient star-style entity search over huge KBs.
- We propose the schema-first entity search algorithm which can efficiently prune clusters of RDF data based on the schema of clusters.
- We test the performance of our algorithms over a huge real-life RDF KB, the billion triple challenge (BTC) [1] data set. The results demonstrate that the efficiency of star-style SPARQL queries can be significantly improved.

The rest of the paper is organized as follows. Section 2 gives the study of related work. Section 3 introduces the schema-first entity search algorithm. Section 4 describes the techniques used for efficient and effective partitioning of the RDF KBs. The experimental study is given in Section 5, followed by the conclusion given in Section 6.

2 Related Work

Some well-known early works on storing, indexing and query processing RDF data include Sesame[4,14], Jena [15], YARS [13] et al. All these systems apply relational tables to store RDF triples. Primary versions of Sesame and Jena maintain all triples in a giant table, which is called triple store. Each query requires a large number of subject-subject self-join. Such a solution thus leads to poor performance. In order to improve the performance, Jena2[21] groups the triples by its property (predicate) name, and store all triples with the same name into the same relational table, denoted as property table. In Jena, three kinds of property tables are designed to fit single-valued property, multi-valued property and their classes. The solution can distinguish the statements of single and multi predicates, so as to locate different queries, however, it can only support simple join queries, and does not achieve good performance for large volume of RDF data.

YARS applies distributed systems to partition triples across multiple machines. Additional contexts are used for each triple to indicate its provenance. [23] stores RDF triples in cloud systems. However, according to their performance evaluation, they only achieve high throughput for certain simple queries that have parallel access patterns to the systems.

In [7], the authors propose to use column store for maintaining RDF triples. It partitions the triple table based on predicates. All the triples of the same predicate are stored in the same vertical table. For triples of the same predicate, the predicate column is removed, which becomes a minimum-width property tables (i.e., binary relations). Authors of [18] map the tables into Monet DB (with column storage architecture) and PostgreSQL (with row storage architecture). They show that better performance is achieved by the column-store solution. [16] deploys a compressed Bit-Matrix structure for storing huge RDF graphs. However, all the four kinds of BitMat proposed in the paper are written in same file, which complicates the update operations, and therefore the solution does not have good scalability.

All these physical designs are complemented by indexes to improve the query efficiency [9,17,18,20]. YARS2 builds 6 indexes of triples in separated B+ tree or hash indexes. Hexastore[20] creates the same number of indexes without any compression. [11] even saves entire paths of the RDF graph and their SPO labels as indexes. These approaches do help to enhance the performance of simple queries. However, due to the lacking of good optimization techniques, they can only support a limited kinds of joins. Moreover, these solutions are not verified on large scale of RDF data.

The state of art RDF query engine, RDF-3X [17], creates single huge triples table to load data. It stores all triples in B+ tree, and build exhausted indexes of all SPO permutations. In order to save space, URIS and literals in the triples are mapped to integer identifiers in RDF-3X. A compress scheme is applied to the indexes. According to the performance experiments, RDF-3X outperforms most of the previous SPARQL query engines in terms of the query processing time. It is widely accepted as the state of art for

SPARQL query processing engine.The author improved the join-order optimization in [10], which make the join process more scalable. However, the performance RDF-3X is still not good enough when a large portion of indexes need to be sequentially scanned. This limits the scalability of RDF-3X when the data grow up to billions. The index scan becomes a bottleneck.

3 Partition-Based Entity Search

3.1 The Entity Search Problem

An RDF triple is simply represented as $t =< s, p, o >$, where $s = Sub(t), p = Pre(t)$ and $o = Val(t)$. A subject s in an RDF triple is an ID/URI that uniquely identifies an entity in the KB. As such, we simply refer an entity whose subject is s as entity s. All RDF tuples of an entity s are denoted as a set $T_s = \cup_{Sub(t)=s}\{t\}$. The predicate set of an entity s is denoted as $P_s = \cup_{t \in T_s}\{Pre(t)\}$, which contains all predicates that the entity s has. We also call P_s as the schema of the entity s. Let the domain of all predicates in an RDF KB be P. We therefore have $P_s \subseteq P$. For a KB containing a large number of various types of entities, we have $|P_s| \ll |P|$.

Entity search over KBs can be defined as to retrieve entities (and their values on some predicates) that satisfying some constraints over some specified predicate values. It is basically a star-style graph pattern matching problem, whose goal is to retrieve entities that perfectly match a specified query pattern. Figure 1 shows an example of entity search. Note that, a general SPARQL query may contain more than one entity variables, e.g., the query pattern given in Figure 3.1. In these cases, a SPARQL query can be addressed by a chain join of the results of multiple entity search queries. This paper is focused on the basic component, star-style entity search query.

```
SELECT ?n ?r ?m
WHERE
{?x name ?n
 ?x research_area Physics
 ?x born_in_place ?r
 ?x awarded Nobel_prize
 ?x affiliation ?y
 ?y located USA
 ?y name ?m}
```

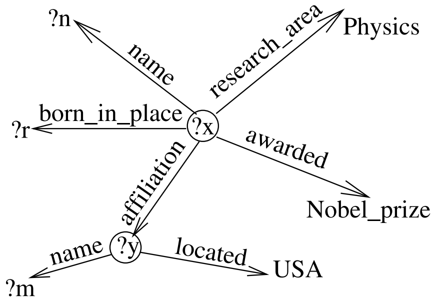


Fig. 2. An example of a general SPARQL query that contains two entity variables. It can be addressed by a chain join of entity search results, according to the BGP $?x$ affiliation $?y$.

3.2 Schema-Valid Entities and Tables

For a query q, let P_q be the set of predicates specified by q. For the example of Figure 1, $P_q = \{name, born\_in\_place, research\_area, awarded\}$. Predicates in P_q can be constrained predicates whose values have been specified (e.g., $research\_area$ and $awarded$ in Figure 1) or wildcard predicates whose values are not specified by the query (e.g., $name$ and $born\_in\_place$ in Figure 1). To match a query q, an entity must contain all predicates of P_q. Moreover, the entity must satisfy all specified values over those constrained predicates of q. An entity s is a *valid entity* of q if it can match to the query pattern specified by q. For example, entity e_1 in Figure 3 is the only valid entity of the query q in Figure 1. The goal of entity search is to retrieve all valid entities to a query q. An entity s is called a *schema-valid* entity of a query q if $P_q \subseteq P_s$. A schema-valid entity of q must contain all predicates in P_q. For example, entity e_1 and e_2 in Figure 3 are schema-valid entities of the query q. Correspondingly, an entity s is a *schema-invalid* entity of a query q if $P_q \nsubseteq P_s$ (e.g., e_3 in Figure 3). Obviously, a schema-valid entity of q may not be a valid entity of q (e.g., e_2). However, to be a valid entity, an entity s must first be a schema-valid entity of the query q. This motivates us to organize RDF entities in a smart way so that we can quickly prune those schema-invalid entities, before looking for valid entities from those schema-valid entities that take minority in the KB.

e1

name	Albert Einstein
born_in_place	Germany
awarded	Nobel_prize
research_area	Physics

e2

name	Jim Gray
born_in_place	U.S.
awarded	Turing_award
research_area	Database

e3

name	Usain Bolt
born_in_place	Jamaica
awarded	Olympic_golden_medal
occupation	Athlete

Fig. 3. Example of entities

In our model, entities are organized in clusters, with each cluster containing a table of entities of similar schemas. We treat a cluster as a virtual table whose schema (property sets) can be dynamically adjusted. A virtual table (hereafter shorted as table) R contains entities whose predicate sets are not necessarily the same. The schema of a table is formulated by merging the schemas of all entities it contains, i.e., $P_R = \cup_{s \in R} P_s$. Obviously, A table R may contain valid entities of q only if $P_q \subseteq P_R$. We say that table

R is *schema-valid* to q if $P_q \subseteq P_R$. In the other way, table R is *schema-invalid* to q if there exists at least one predicate p such that $p \in P_q$ and $p \notin P_R$. It is guaranteed that there are no valid entities of q in table R that is *schema-invalid* to q. We therefore can prune those *schema-invalid* tables of q when processing the query q.

3.3 Schema-First Entity Search

Based on the table's schema, we can quickly identify those *schema-valid* tables of a given query. This is achieved by creating inverted lists of tables for all predicates. Each predicate $p \in P$ has an inverted list l_p recording all the tables that having p as one predicate in their schema. The tables in an inverted list are sorted and indexed. Because the number of tables and predicates are very limited, the inverted lists can be easily held in memory. Given a query q whose specified predicate set is P_q, all *schema-valid* tables to q can be efficiently discovered by scanning the inverted lists of all predicates of P_q. A sorted merge process will be applied to efficiently retrieve those *schema-valid* tables from the relevant inverted lists.

A *schema-valid* table of a query does not have to contain valid entities of the query. An entity s in a *schema-valid* table may not match the query pattern of q in two ways: (1) there exists one predicate $p \in P_q$ that $p \notin P_s$; (2) some predicate values of s do not satisfy the values specified by the constrained predicates of q. A refining process is therefore required for retrieving valid entities from a *schema-valid* table of q. Note that the number of entities is often huge for large KBs. They therefore need to be maintained in external devices. Indexing techniques can be applied for efficient retrieval of valid entities from virtual tables. In our study, we apply the state of art, RDF-3X, for indexing entities of one virtual table. They are therefore called clustered indexes. Compared to building SPO indexes over all the RDF triples, the way of building clustered indexes over small virtual tables generates smaller predicate-ahead indexes. It therefore saves the cost of scanning predicate-ahead indexes. This is the reason why clustered indexes are able to improve the performance of entity search queries conducted over the indexes of the whole data set. Finally, all valid entities extracted from those *schema-valid* tables

Algorithm 1. Schema first entity search (SFES)

```
Input: $q$: a given query for searching entities in KB
Output: $S$: the set of valid entities of $q$
retrieve the set $R_q$ of all schema-valid tables of $q$, by merging the inverted lists of all predicates of $P_q$;
for each table $R \in R_q$ do
    process $q$ as a SPARQL query using the RDF-3X techniques, with indexes built over entities of $R$;
    let $S_R$ be the result set of the above query processing over virtual table $R$, i.e., the set of valid entities of $q$ in $R$;
    $S = S \cup S_R$;
return $S$;
```

of a given query q will be combined, as the final results of the entity search query q. We call the whole process of the above entity search algorithm as the Schema-First Entity

Search (SFES) algorithm, which is described in Algorithm 1. Schema-first entity search can quickly identify a small number of virtual tables so that the expensive entity search computation can be paid to those entities that are more likely relevant to the query.

4 Efficient and Effective Entity Partitioning

The performance of the SFES algorithm will be highly dependent on how entities are allocated across virtual tables. According to the SFES algorithm, two factors are important for achieving efficient schema-first entity search. First, there should be a small number of virtual tables *schema-valid* to a given query because a relation query process will be conducted for each *schema-valid* table. Second, for each *schema-valid* table, a large percentage of its entities should also be *schema-valid* to the query. Those *schema-invalid* entities will generate extra I/Os when loading indexes for some predicates (contained by those *schema-invalid* entities) of the table.

The schema of a table R is a superset of the schemas of entities it contains. As a result, the more entities a table R has, the more predicates it probably contains. In one extreme case, if all entities in the KB are maintained by only one table R, it will be a wide table solution where $P_R = P$. In the wide table solution, each entity has empty values on most predicates because $|P_s| \ll |P_R|$ in this case. The wide table R will be *schema-valid* to any query q (assuming only predicates in P are used by the query q). In the other extreme case, if only entities of the same schema share a table R, any entity s in the table R will have values on any predicate of P_R because $P_s = P_R$. However, in this case, a huge number of tables will be generated because there are always no pre-defined schemas for entities to conform to, in a KB.

In our solution, an entity exists only in one table of the KB. The important problem is how to allocate entities to virtual tables. A reasonable solution of allocating entities should be in-between the above two extreme cases. Each table contains a number of entities of similar schemas so that, (1) the number of entities contained by each table is significant enough such that more entities can be pruned when a *schema-invalid* table is pruned; (2) the number of predicates for the schema of each table should be not large (compared to the number of predicates each entity has), so that entities in a *schema-valid* tables are likely to be also *schema-valid*. The above two goals are self-contradictory. The more entities a table contains, the larger the schema size of the table. An elaborate solution of allocating entities into clusters is therefore desired to achieve a good trade-off between the above two goals.

4.1 Entity Clustering

We observe that many entities of similar types have very similar schemas in existing RDF KBs. For examples, movie entities extracted from IMDB, paper entities extracted from DBLP. Those well-formed entities usually hold the majority in the KB. Based on this observation, we extract tables from KB by clustering those entities of similar schemas together. Considering that some entities are "outliers" in terms of the schemas they have, they are maintained by a separated virtual table $\bar{R}$, called the fusion table. The system therefore maintains a number of tables $R_1, R_2, \ldots, R_n$, and the fusion table $\bar{R}$. Each entity in the KB must be in one of the tables.

An optimal solution of allocating RDF entities is that, given a query q, we can find a table R that exactly contain all *schema-valid* entities of q. There are no *schema-invalid* entities of q in table R, which means all entities in R share the same schema. However, this is usually impractical because many different schemas of entities often significantly overlap with each other. When the predicate set P_q of a query is a subset of the intersection of multiple schemas, the tables of those schemas will be all *schema-valid* to the query q.

Clustering of entities is necessary to allocate entities of similar schemas into clusters. We can define the distance of two entities based on the similarities of their schemas. The Jaccard distance obviously serves this purpose. Given two schemas P_1 and P_2, the Jaccard distance is defined as the $d(P_1, P_2) = \frac{|P_1 \cup P_2| - |P_1 \cap P_2|}{|P_1 \cup P_2|}$. There are a number of clustering approaches, e.g., k-medoids [19] that can be applied for clustering of entities. Considering the number of entities in a KB can be as large as billions. It will be extremely expensive to run common clustering algorithms over such a huge data set of RDF entities.

Taking the idea of Dirichlet process [22], we design an incremental clustering algorithm (Algorithm 2) which can efficiently cluster entities simply by scanning the whole dataset for once. In our clustering algorithm, we first aggregate entities of the same schemas so that we can cluster directly over schemas, instead of entities (whose number is much larger than that of schemas). All the schemas are ranked based on the numbers of entities they have. Then, a one-pass scanning process is conducted over the ranked schema list. Schemas with large population are processed first.

The incremental clustering algorithm shown in Algorithm 2 works as follows. The first schema forms a cluster directly. For each following newly scanned schema P, it will be compared against all the existing clusters. Schema P can be allocated to an existing cluster R only if 1) $P \subset P_R$; 2) $d(P, P_R)$ is no more than the distances of P to any the other cluster; 3) $d(P, P_R)$ is no more than some given threshold ε. If the schema P cannot be allocated to any existing cluster, a new cluster will be created to hold it. Finally, all schemas are allocated into a large number of clusters. Each head clusters with enough entities is treated as a table. In contrast, those tailed clusters with few entities will be merged into the fusion table $\bar{R}$.

The k-medoids algorithm achieves the optimal clustering strategy that requires to scan the data set for multiple times to guarantee the convergence of the partitions. For each iteration, the re-computation of medoids requires a complexity of $O(\frac{n^2}{k})$, which is also very expensive. Comparatively, the above incremental clustering algorithm only need to scan the whole dataset for once. The medoid of each cluster is fixed as the first schema that initializes the cluster. It is therefore the schema of the whole cluster. Because schemas of large population are processed first, the origin schema of a cluster therefore usually takes majority of entities in that cluster. The predicate sets of the following schemas allocated to a cluster are subsets of the origin schema. As for the above reasons, the incremental clustering algorithm (Algorithm 2) is much more efficient than traditional clustering algorithms such as k-medoids.

Although algorithm 2 may not achieve optimal partitioning strategy that can be achieved by the k-medoids algorithm. It however does not need to specify the parameter k beforehand. The quality of partitions is mainly controlled by the parameter ε, which

Algorithm 2. Entity partitioning

```
Input: $P_1, \ldots, P_t$, a list of rankded schemas.
Input: $\varepsilon$, maximal distance for merging schemas.
Input: $\delta$, minimal entities that a table at least have.
Output: $R_1, \ldots, R_n, \bar{R}$, tables for entities.
$\mathcal{R} = \emptyset$, set of existing clusters.
for each schema $P$ do
    $minD = +\infty, minR = -1$;
    for each cluster $R_i \in \mathcal{R}$ do
        if $P \subseteq P_{R_i}$ then
            $D = d(P, R_i)$;
            if $D < minD$ and $D < \varepsilon$ then
                $minD = D$;
                $minR = i$;
    if $minR \geq 0$ then
        allocate $P$ to cluster $R_i$;
    else
        create a new cluster for $P$ and insert it into $\mathcal{R}$;
$\bar{R} = \emptyset$;
for each cluster $R_i \in \mathcal{R}$ do
    if $|R_i| < \delta$ then
        merge $R_i$ to $\bar{R}$;
    else
        treat $R_i$ as a table;
```

determines the similarities between the subsequent schemas and the origin schema. The smaller the ε, the more similar of different schemas in each cluster, the larger number of clusters will be generated. Besides ε, the parameter δ also help to control the quality of a cluster and the number of overall clusters. This is because, after the partitioning process, all the clusters whose number of entities less than δ will be merged into the fusion table $\bar{R}$. By properly setting ε and δ, the fusion table $\bar{R}$ will be much smaller than the original dataset because in real-life RDF KBs, most entities have a large number of isomorphic entities which have the same schema.

4.2 Updates of Partitioned Indexes

Most studies for indexing RDF KBs does not consider the update problem. Many of them follow an assumption that the indexes can be re-built from scratch when enough updates occur. However, this assumption has a problem that the newly updated RDF triples will not be indexed timely and therefore may not be search before index updates. In our study, we only consider the insertion of new triples as update operations, i.e., the RDF KBs monotonously grow up.

The clusters generated from Algorithm 2 are static analysis of the KBs. However, with the enlargement of KB, the original partitions may not be effective enough. The clustering process is very expensive. It cannot be trigger frequently due to the frequent updates of KBs. To address this, we utilize the fusion table $\bar{R}$ as a buffer for caching newly updated entities and triples. As shown in Algorithm 3, the updates of entities and triples are respectively processed. Firstly, when an entity is inserted, we will check its schema. If entities of the same schema have been allocated to one particular partition, the new entity will be inserted into that partition, which further triggers the index updates of that cluster. If the schema of the new entity has not been allocated to any partition, it will be cached and indexed by the fusion table $\bar{R}$. Secondly, when a triple is inserted, we will check the schema of the subject of that triple. If it is in one partition, we will check whether the schema of the updated entity still belongs to the schema of that partition. If it is, the triple will be inserted to that cluster. Otherwise, the entity will be removed from that cluster, and inserted as a new entity to the system.

Algorithm 3. Updates of partitioned index

```
updateEntity($e$):
$e$, the inserted entity;
if $P_e$ has been assigned to a virtual table $R_i$ then
    assign $e$ to $R_i$, and update the indexes of $R_i$;
else
    assign $e$ to $\bar{R}$, and update the indexes of $\bar{R}$;

updateTriple($t$):
$t$, the inserted triple;
let $e = S_t$, the entity of the triple;
if $e$ has been assigned to a virtual table $R_i$ then
    update $P_e$;
    if $P_e \subseteq P_{R_i}$ then
        insert $t$ into $R_i$, and update its indexes;
    else
        remove $e$ from $R_i$;
        updateEntity($e$);
else
    assign $t$ to $\bar{R}$, and update its indexes;
```

Such a way of partition and index updates always guarantees the correctness of the SFES algorithm. However, with the increment of the updates, the fusion table $\bar{R}$ will grow up. We therefore periodically check the schemas in $\bar{R}$. They will be treated as the input of the clustering algorithm (Algorithm 2) when updates are triggered. In this way, some schemas of $\bar{R}$ can be inserted into some existing partitions; some schemas of $\bar{R}$ can generated new partitions if they have enough population (no less than δ) and cannot be assigned to any existing partitions. As such, the partitions and fusion table $\bar{R}$ are incrementally maintained.

5 Experimental Study

5.1 Experimental Setup

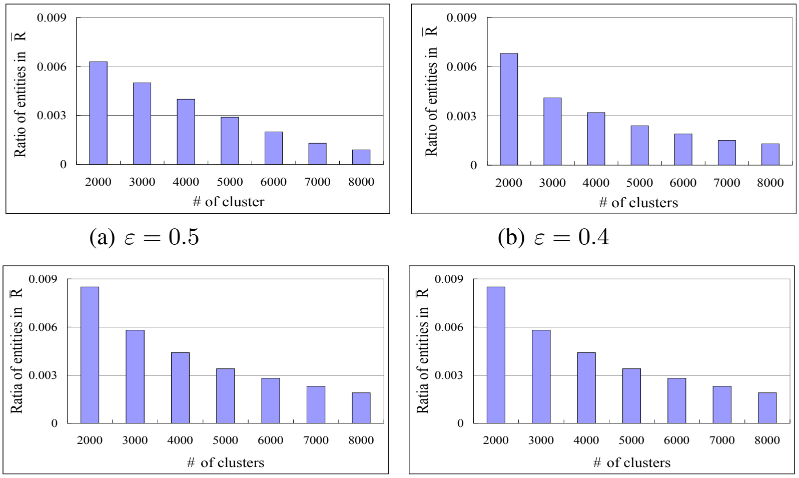
We use C++ to implement our algorithm and run the experiments on a PC with a 1.8Ghz Core 4 Duo processor, 10 GB memory and running 64-bit Linux Redhat kernel.

We choose two baselines as competitors of our solution: one uses RDF-3X version 0.3.5 for indexing and querying RDF triples; the other uses PostgreSQL 8.4.1 as a triple store with indexes over subjects supported. In RDF-3X, URIs and literals of RDF triples are mapped into integers. The indexes are maintained by B+ tree. It takes around 5 hours to index all triples in our experiments by RDF-3X engine. We observe that RDF-3X takes significant amount of time on converting IDs to Strings.

Billion Triple Challenge 2010 (BTC) [1] is a dataset with more than 3.2 billion triples. It contains triples from twelve sources such as Yago, DBpedia, Freebase. We choose BTC2010 as the experimental dataset so that our solution can be evaluated on large scale of real-life RDF data. The dataset has 95,898 distinct predicates. The original data format in the dataset is the NQuads format which includes a subject, a predicate, an object and a context for each triple. We omit some noisy data and the context, thus kept 1,026,823,962 triples (after de-duplication) for evaluation. The space consumption for RDF-3X indexes and PostgreSQL triple store are 47GB and 125GB respectively.

5.2 Experiments on Analysis Clustering Results

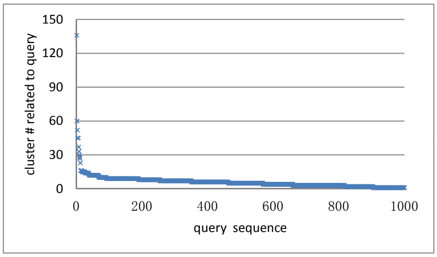
The first experiment is to test the impact of parameters in Algorithm 2 on the ratio of entities falling in the fusion table $\bar{R}$. Obviously, the less the ratio of entities falling in $\bar{R}$, the more entities that have been clustered in virtual tables. We set parameter ε as $\varepsilon = 0.5$, 0.4, 0.3 and 0.2 respectively. By varying the parameter δ, we are able to generate the number of clusters from 2000, 3000,..., to 8000. The results are shown in Figure 4. From the results, we can see that the ratios of entities in $\bar{R}$ are very small (less than 0.01) in all the cases. When fixing the parameter ε and increasing the number of clusters generated, the ratio of entities in $\bar{R}$ drops. On the other way, when the number of clusters is fixed (e.g., 8000) and dropping the parameter ε, we can find that the less the ε, the more the ratio of entities in $\bar{R}$. This is because small ε guarantees the entities within a cluster are more similar with each other. In our solution, we choose as $\varepsilon = 0.2$ and the number of clusters as 2000. We also test the query hits of clusters (*schema-valid* tables) for the partitioned indexes by using random generated queries. The query patterns are randomly generated based on the following steps: 1), randomly pick an entity from the RDF dataset; 2), randomly select a number of predicates (no more than 5) from the schema of the picked entity of step one, forming the query schema P_q; 3), count the number of clusters that are schema-valid to the query. In step two, we have two alternatives: one is evenly select a number of predicates from the schema of the entity; the other is to select predicates based on the frequency of that predicate in the whole dataset (therefore popular predicates have a larger probability to be picked). For each of the alternatives, we generate 500 queries. The distribution of the number of query hits for the 1000 queries is plotted in Figure 5. The results show that for most of queries, no more than 10 clusters are *schema-valid* queries. The majority of the clusters can therefore be pruned without further query processing. We also did some statistics to show the distribution of each predicate over the number of clusters, that is to compute frequency of each predicate in clusters (shown in Figure 6). The figure indicates that most predicates are contained by less than 20 clusters.



(a) $\varepsilon = 0.5$ (b) $\varepsilon = 0.4$

(c) $\varepsilon = 0.3$ (d) $\varepsilon = 0.2$

Fig. 4. The ratio of entities in the fusion table $\bar{R}$ when varying the cluster numbers and ε



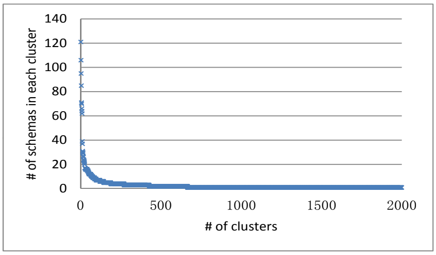


Fig. 5. The plot of query hits for randomly generated queries

Fig. 6. The plot of the number of clusters containing a predicate

5.3 Query on BTC

We use 10 queries to compare the performance of our solution with the baselines. They are listed in Appendix A (Q1-Q10). For PostgresSQL, before submitting to the query engine, the queries are manually rewritten into SQL clauses. The performance of PostgreSQL is the worst among the three systems. The queries can be divided into the following three categories; the first category is star join queries with popular predicates and unspecified object (Q1, Q4, Q6, Q10); the second category is star join queries with unpopular predicates (Q7, Q8); the third category of queries are the queries with unknown predicates (Q2, Q3, Q5, Q9). For each query, it first looks up in the inverted list to find the relevant table. Although the time consumption in this filter step is much more less than search in tables, we still count it in the query time. The results are shown in Table 1 and Table 2.

For the first category of queries, our solution outperforms the other two solutions. As discussed, when a query contains a popular predicate (e.g., <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> in Q1.) with its object unspecified (variable), the RDF-3X engine need to load the whole predicate-ahead index of the popular predicate, which is typically very large in volume. However, for the partitioned indexes, our solution can prune all the *schema-invalid* clusters. While, the indexes of popular predicates will not be very large because instances (entities) of the popular predicates have been de-clustered into many partitions. Only those of *schema-valid* tables are loaded. For the second category of queries, the performance of our solution achieved a little over RDF-3X. In our solution the unpopular predicates usually relevant to more clusters, so more *schema-valid* tables are loaded for allocating the results. In the third category, although there are unknown predicates in the queries, our solution prune clusters by using the rest known predicates in first step. After that the system can filter the unknown predicate based on its object and results of the first step. The result shown that with only one popular predicate in the rest part of query(as shown in Figure 5,it is the most cases), our solution still has better performance than the competitor.

Table 1. Query run-times in seconds for the Billion Triples Challenge dataset: Q1-Q6

	Q1	Q2	Q3	Q4	Q5	Q6
cold caches						
Our Solution	0.1	0.05	0.4	3.16	0.12	0.05
RDF-3X	1.38	0.1	0.46	57.76	0.38	0.29
PostgreSQL	$> 30min$	124.63	$> 30min$	$> 30min$	711.6	$> 30min$
warm caches						
Our Solution	0.003	0.009	0.02	2.63	0.012	0.01
RDF-3X	0.03	0.017	0.035	9.69	0.019	0.03
PostgreSQL	$> 30min$	98.31	$> 30min$	$> 30min$	680	$> 30min$

Table 2. Query run-times in seconds for the Billion Triples Challenge dataset: Q7-Q10

	Q7	Q8	Q9	Q10	mean (Q1-10)
cold caches					
Our Solution	1.15	0.63	0.2	0.09	0.4
RDF-3X	1.15	0.65	0.82	0.38	0.86
PostgreSQL	1715	1600	$> 30min$	1218	> 80.66
warm caches					
Our Solution	0.2	0.08	0.013	0.009	0.099
RDF-3X	0.2	0.08	0.024	0.019	0.16
PostgreSQL	1510	1511	$> 30min$	1033	> 75.31

6 Conclusion

In this study, we show that the performance of star-style SPARQL join queries (or entity search queries) in huge RDF knowledge bases can be improved by effectively partitioning entities into clusters based on their schemas. We design an efficient and effective clustering algorithm for partitioning the entities in huge RDF knowledge bases. The experimental studies over huge real-life RDF dataset demonstrate the pruning power of the SFES algorithm. It therefore can save the cost for loading large indexes when processing entity search queries where the combination of predicates can be applied for filtering entity partitions.

Acknowledgements. This work is funded by the National Science Foundation of China under Grant No. 60873017 and No. 61170010.

References

1. Billion Triple Challenge, http://challenge.semanticweb.org/
2. Freebase, http://www.freebase.com/
3. Linked Open Data, http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
4. OpenRDF, http://www.openrdf.org/index.jsp
5. SPARQL, http://www.w3.org/TR/rdf-sparql-query/
6. YAGO, http://www.mpi-inf.mpg.de/yago-naga/yago/
7. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.J.: Scalable Semantic Web Data Management Using Vertical Partitioning. In: 33th International Conference on Very Large Data Bases, pp. 411–422 (2007)
8. Bizer, C., Heath, T., Idehen, K., Berners-Lee, T.: Linked data on the web (LDOW 2008). In: 17th International Conference on World Wide Web, pp. 1265–1266 (2008)
9. Bröcheler, M., Pugliese, A., Subrahmanian, V.S.: DOGMA: A Disk-Oriented Graph Matching Algorithm for RDF Databases. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 97–113. Springer, Heidelberg (2009)
10. Neumann, T., Weikum, G.: Scalable join processing on very large RDF graphs. In: 2008 ACM SIGMOD International Conference on Management of Data, pp. 627–640 (2009)
11. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge Unifying WordNet and Wikipedia. In: 16th International Conference on World Wide Web, pp. 697–706 (2007)
12. Zemánek, J., Schenk, S.: Optimizing SPARQL Queries over Disparate RDF Data Sources through Distributed Semi-Joins. In: International Semantic Web Conference (Posters & Demos) (2008)
13. Harth, A., Decker, S.: Optimized Index Structures for Querying RDF from the Web. In: LA-WEB, pp. 71–80 (2005)
14. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: An architecture for storing and querying rdf data and schema information. Web Sem. 8(4), 271–277 (2010)
15. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: Implementing the Semantic Web Recommendations. In: WWW Alt, pp. 74–83 (2004)
16. Atre, M., Chaoji, V., Zaki, M.J., Hendler, J.A.: Matrix "Bit" loaded: a scalable lightweight join query processor for RDF data. In: 19th International Conference on World Wide Web, pp. 41–50 (2010)
17. Neumann, T., Weikum, G.: RDF-3X: a RISC-style engine for RDF. PVLDB 1(1) (2008)
18. Sidirourgos, L., Goncalves, R., Kersten, M.L., Nes, N., Manegold, S.: Column-store support for RDF data management: not all swans are white. PVLDB 1(2), 1553–1563 (2008)
19. Theodoridis, S., Koutroumbas, K.: Pattern Recognition, 3rd edn. Academic Press, Inc., Orlando (2006)

20. Weiss, C., Karras, P., Bernstein, A.: Hexastore: sextuple indexing for semantic web data management. PVLDB 1(1), 1008–1019 (2008)
21. Wilkinson, K., Sayers, C., Kuno, H.A., Reynolds, D.: Efficient RDF Storage and Retrieval in Jena2. In: SWDB, pp. 131–150 (2003)
22. Zhang, Z., Dai, G., Jordan, M.I.: Matrix-Variate Dirichlet Process Mixture Models. Journal of Machine Learning Research - Proceedings Track 9, 980–987 (2010)
23. Stein, R., Zacharias, V.: RDF On Cloud Number Nine. In: Proceedings of the 4th Workshop on New Forms of Reasoning for the Semantic Web: Scalable & Dynamic, CEUR Workshop Proceedings, pp. 11–23 (May 2010)

A Query List Used in Experiments

Prefix list: dbpedia: <http://dbpedia.org/property/>
dbpedias:<http://dbpedia.org/resource/>
geo: <http://www.geonames.org/>
pos:<http://www.w3.org/2003/01/geo/wgs84_pos#>
ontology: <http://dbpedia.org/ontology/>
skos: <http://www.w3.org/2004/02/skos/core#>
dmdb: <http://data.linkedmdb.org/resource/movie/>
purl: <http://purl.oclc.org/NET/nknouf/ns/bibtex#>
daml: <http://www.daml.org/2003/02/fips55/>
semweb: <http://www.cs.cas.cz/semweb#>

Q1: select ?a ?type ?pub where { ?a <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?type. ?a semweb:publisher ?pub. ?a semweb:periodical_title ”Theory of Computing Systems”. }

Q2: select distinct ?a ?lat ?long ?pop where { ?a [] ”Chevilly”. ?a geo:ontology#in Country geo:countries#FR. ?a pos:lat ?lat. ?a pos:long ?long. ?a geo:ontology# population ?pop.}

Q3: select distinct ?l ?long ?lat where { ?a [] ”Luciano Emilio en”. ?a dbpedia:placeOf Birth ?l. ?l pos:lat ?lat. ?l pos:long ?long. }

Q4: select ?x ?y where {?x <http://www.w3.org/2000/01/rdf-schema#label> ?y. ?x dmdb:director dmdb:director/3480. }

Q5: select ?name ?lat ?long where { ?a [] dbpedias:List_of_World_Heritage_Sites_in_ Europe . ?a dbpedia:name ?name. ?a pos:lat ?lat. ?a pos:long ?long. ?a skos:subject dbpedias:Category:Ancient_Greek_cities. }

Q6: select ?x ?author ?title where { ?x purl:hasAuthor ?author. ?x purl:hasBooktitle ”ISWC 2009”. ?x purl:hasTitle ?title. }

Q7: select ?a ?name ?loc ?postcode where ?a daml:fips-55-ont#name daml:NY.owl#NY. ?a daml:fips-55-ont#directlyLocatedIn ?loc. ?a daml:fips-55-ont#postcode ?postcode.

Q8: select ?x1 ?hometown where { ?x1 ontology:birthPlace ”Chile”. ?x1 ontology:death Place ?death. }

Q9: select ?a ?name ?bn where { ?a [] ?name. ?a dbpedia:placeOfBirth ”Brooklyn, New York en”. ?a dbpedia:dateOfBirth ?bn. }

Q10: select distinct ?name ?lat ?long ?pop where { ?a dbpedia:name ?name. ?a dbpedia:regoin dbpedias:Andaman_Islands. a pos:lat ?lat. ?a pos:long ?long. ?a dbpedia:population ?pop. }

SINBAD: Towards Structure-Independent Querying of Common Neighbors in XML Databases

Ba Quan Truong[1], Sourav S. Bhowmick[1], and Curtis Dyreson[2]

[1] School of Computer Engineering, Nanyang Technological University, Singapore
[2] Department of Computer Science, Utah State University, USA
{bqtruong,assourav}@ntu.edu.sg, curtis.dyreson@usu.edu

Abstract. XML query languages use *directional* path expressions to locate data in an XML data collection. They are tightly coupled to the structure of a data collection, and can fail when evaluated on the *same data* in a *different structure*. This paper extends XPath expressions with a new structure-independent, non-directional axis called the *neighborhood* axis. Given a pair of context nodes, the *neighborhood* axis returns those nodes that are *common* neighbors of the context nodes in *any* direction. Such axis finds its usefulness in structure-independent query formulation as well as supporting relevant results computation in design-independent XML keyword search. We propose an algorithm called SINBAD that exploits the novel notion of *node locality* and small size of XML *structure tree* to efficiently determine the common neighbors of the context nodes. Our empirical study demonstrates that SINBAD, built on top of an existing *path materialization*-based relational storage scheme, has promising query performance.

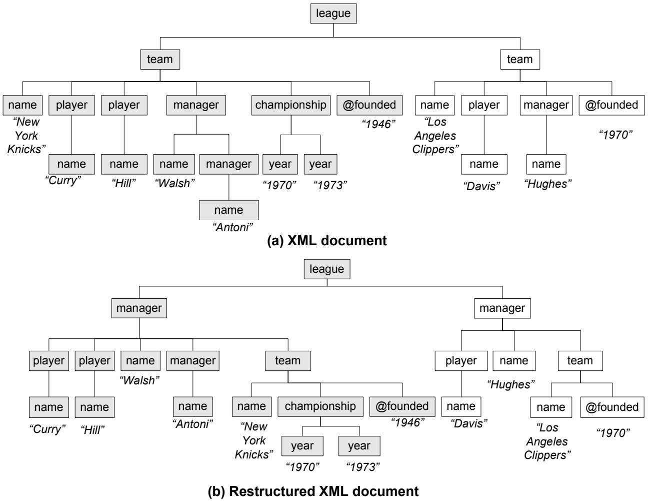
1 Introduction

A wealth of existing literature has extensively studied evaluation of various navigational axes in XPath expressions [6]. A key common feature of these axes is that they are all *directional* in nature. That is, they locate nodes in a fixed direction relative to a context node (*e.g.,* the descendent axis corresponds to the "down" direction). Unfortunately, queries that rely on directional axes become dependent on the data being in the specified direction, even though data has no "natural" direction and can be organized in different hierarchies. Users who are unfamiliar with a document structure or are knowledgeable about a structure which subsequently changes will sometimes formulate *unsatisfiable queries*, which are queries that fail to produce desired results. These queries are difficult to debug since they run to completion and produce a result, though not the desired one.

As an example, consider the XML document in Figure 1(a). It contains league information organized by teams. Each team consists of a set of players. Suppose that a basketball commentator, John, wishes to find the *common* team of a player, *Hill*, and a manager, *Antoni*. John can issue any one of the following XPath queries to retrieve desired information: Q_1: `//player[name='Hill']/ancestor::team[/descendant ::manager[name='Antoni']]` or Q_2: `//manager[name='Antoni']/ancestor::team[/descendant::player[name='Hill']]`.

To correctly formulate the aforementioned queries, John has to know something about the hierarchical structure of the XML data. For instance, he must know that a

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 156–171, 2012.
© Springer-Verlag Berlin Heidelberg 2012



(a) XML document

(b) Restructured XML document

Fig. 1. XML documents

team element is an ancestor of player and manager elements. Furthermore, the team subtree also includes information related to the team. But if John misunderstands the structure or if the structure changes over time then this partial knowledge may not be useful anymore for formulating satisfiable queries as demonstrated below.

Assume that the XML document in Figure 1(a) is now reorganized to the structure depicted in Figure 1(b). Specifically, the document in Figure 1(b) has *same data* but the *structural relationships* of the elements are different. Now the league information is organized according to head managers instead of teams. Both documents contain the same data and same element labels but they have different hierarchical relationships. Such structural change is real because database administrators may revise the design over time to address issues such as redundancy, space overhead, performance, and usability [4,12]. Unfortunately, due to the lack of non-directional axes in XPath, for some queries different path expressions are needed to query each hierarchy. Consequently, Q_1 and Q_2 may become unsatisfiable on the document in Figure 1(b) as the team element is no more an ancestor of player or manager elements.

Note that it is unrealistic to expect users to be "structure-aware" as it does not scale with increasing structural heterogeneity. Is it possible to retrieve the above information using a single query without being aware of the underlying structural heterogeneities of elements? Ideally, such a query technique should work even if the document structure is reorganized. In order to answer this question affirmatively, in this paper we propose a new *non-directional* XPath axis called neighborhood axis, which enables us to locates all *common nodes* of two context nodes in *any* direction.

Specifically, the XPath language is extended with a *non-directional locator*, called the neighborhood axis, to support non-directional exploitation of XML data. The proposed axis allows a user to formulate precise queries knowing only the labels of nodes and unaware of the exact hierarchy. Informally, given *two* context nodes, the neighborhood axis returns those nodes that are common nodes to these context nodes. For example, reconsider the query posed by John. The relevant team node must be related to both the player node containing *Hill* and the manager node containing *Antoni*. Accordingly, John can reformulate his query using the neighborhood axis as follows: Q_3: //player[name ='Hill']/neighborhood{//manager[name ='Antoni']}::team.

Note that Q_3 will retrieve the same information when it is evaluated over Figure 1(b) as well. More importantly, a user does not need to be aware of the structural relationship between the context and test nodes. In this case, John only needs to know that a team could employ a player and a manager (real-world employment relationship). He does not need to know the relative hierarchical relationship among them (e.g., team is ancestor or descendant of manager) in the document.

The neighborhood axis has practical significance in at least two applications. Firstly, it can complement classical approach to query XML data by enabling users to formulate *structure-independent* queries to seek common nodes of a pair of context nodes. Note that classical XPath axes fail to formulate such structure-independent queries. Secondly, it can provide a framework to support *design-independent* XML keyword search [11] by finding relevant nodes that are semantically related to a set of nodes containing matching query keywords. These nodes can be returned with the result set in order to ensure that the results of XML keyword search are informative.

We propose a novel and generic algorithm called SINBAD (**S**tructure **I**ndependent commo**N** neigh**B**ors **A**bstraction proce**D**ure) to evaluate neighborhood axis by exploiting the notion of *node locality*. Informally, given a context node c, the *locality* of c is a set of nodes that are *semantically related* to c (detailed in Section 3). The intuition behind node locality is that users (queries) are typically interested in nodes within the locality and rarely refer to nodes outside of the locality. As we shall see later, the evaluation of neighborhood axis is equivalent to finding the *intersection region* of two node localities.

In summary, this paper makes three main contributions. First, we extend classical XPath query language with a non-directional neighborhood axis in Section 4. Secondly, in Section 5 we present a novel algorithm called SINBAD to evaluate neighborhood axis queries by exploiting the notion of node locality. Thirdly, through an experimental study on synthetic and real data sets, in Section 6, we show that our approach can retrieve common neighbors efficiently .

2 Related Work

Our objective to flexibly issue XML queries independent of the structure is shared by several recent papers. [3] presents a semantic search engine for XML. The search relies on an interconnection relationship to decide whether nodes are semantically related. Two nodes are interconnected if and only if the path between them contains no other node that has the same label as the two nodes. [7] proposes a schema-free XQuery, facilitated by a *Meaningful Lowest Common Ancestor Structure* (MLCAS) operation. Unlike neighborhood axis, these approaches do not retrieve common neighbors of two context nodes.

Recently, several XML keyword search techniques [8,9,13] have been proposed to offer more user-friendly solution for retrieving relevant results. Essentially, these approaches return variants of the subtree rooted at the lowest common ancestor (*e.g.,* VLCA, SLCA) of all the keywords. Due to the lack of expressivity and inherent ambiguity of keyword search, several techniques have also been developed to infer and retrieve relevant results for a search query [8,9,11]. Our work differs from the keyword search paradigm in the following ways. First, we retrieve nodes based on common locality of a pair of context nodes and not the entire LCA-variant of all the keywords. Note that LCA and its variants make use of some common ancestors of the context nodes and therefore rely on the hierarchical relationships. Consequently, these techniques are not structure-independent. Secondly, as a neighborhood query is an extension of conventional XPath query, it can impose more complex predicates compared to keyword search queries. Furthermore, it does not suffer from expressivity and ambiguity issues similar to keyword search.

More germane to this work is our previous efforts in [1,15]. In [15], we extended the XPath language with a *symmetric* locator, called the `closest` axis, which locates nodes that are *closest* to a context node. Here *closest* is measured by the distance from the context node in *any* direction in the XML tree. In [1], we proposed `rank-distance` axis, which is a more generic non-directional axis compared to the closest axis. Specifically, given a context node and two parameters α and β, the `rank-distance` axis returns those nodes that are ranked between α and β in terms of closeness from the context node. Not only it can find closest node(s) (by setting α and β to one) but also nodes that are further away from the context node. In contrast, here we focus on a new axis, called `neighborhood`, which computes common neighbors of two context nodes.

Note that common neighbors cannot be computed using closest axis. For example, reconsider the query in Section 1 for finding the common team of *Hill* and *Antoni*. At first glance, it may seem that this query may be expressed as follows: Q_4: `//player[name='Hill']/closest::team[closest::manager[name='Antoni']]`. Unfortunately, Q_4 returns empty result set. The fragment `//player[name='Hill']/closest::team` will return the team of Hill (which is *New York Knicks*). Hence, when the context node is at this `team` node, the closest `manager` node is, unfortunately, not *Antoni* but manager *Walsh*. Note that we cannot use `rank-distance` axis to select *Antoni* here as it demands knowledge of structural relationship between manager nodes (*Antoni* is a second-level manager) from the user in order to assign appropriate values to the parameters α and β.

3 Node Locality

In this section, we introduce the notion of *node locality* that we shall be using to define neighborhood axis. We begin by briefly introducing the XML data model considered in this paper.

We model XML documents as ordered, labeled trees as follows. A tree is a tuple $(\mathcal{N}, \mathcal{E}, \Sigma, \mathbb{L}, \mathbb{F}, \mathbb{T}, \mathcal{S})$, where (a) $\mathcal{N}$ is the node set. $r \in \mathcal{N}$ is a special node called the root of the tree, (b) let O be the domain of ordinals. Then $\mathcal{E} \subseteq O \times \mathcal{N} \times \mathcal{N}$ is the edge set such that (i) each edge has an ordinal $o_i \in O$ to represent ordering among the children; (ii) there is a path between every pair of nodes; (iii) there is no cycle among the edges; and (iv) every edge has a single incoming edge, except r, which has no incoming edge, (c) Σ is an *alphabet* of labels and text values, (d) $\mathbb{L} : \mathcal{N} \rightarrow \Sigma$ is a *label function* that maps each node to its label, (e) $\mathbb{F} : \mathcal{N} \rightarrow \Sigma \cup \{\epsilon\}$ is a value function that maps a node to its value, in which $\mathbb{F}(n) = \epsilon$ if node n has an empty value, and (f) $\mathbb{T} : \mathcal{N} \rightarrow \mathcal{S}$ is a *type function* that maps each node to a *type* within the *type set* $\mathcal{S}$.

This simple model, which is sufficient for this paper, ignores comments, attributes, processing instructions and namespaces. The model distinguishes between *labels* and *types*. The *label function* maps each node to its label, that is, its element tag. The *type* function specifies the type of each node, where two nodes with the same label could have different types. The type could be defined in various ways, we assume only that each node has a known type. In this paper, the type of a node $n \in \mathcal{N}$ is defined as the *root-to-node path* of n (*i.e.,* the concatenation of the labels on the path from the root to n). For example, suppose that there exist `name` nodes in subtrees rooted at `team` and `player` nodes. Then the path from the root to a team `name` node and a player `name` node differs; therefore they are of different types.

3.1 Intuition

Given a context node, the *node locality* (locality for brevity) is the set of nodes that are *semantically related* to the context node. A node within the locality is called a *local node*. For example, the filled nodes in Figure 1(a) depict the locality of the first `team` node (*New York Knicks*). For instance, the `league` node describes Knicks' league. The two `player` nodes are Knicks' players. The `name` node *Walsh* is Knicks' head manager. Notably, the context node itself is also within the locality.

A key characteristic of node locality is that it is *structure-independent*. That is, when the document structure changes[1], the locality does not change. For instance, all local nodes of team *New York Knicks* in Figure 1(a) are also local nodes of *New York Knicks* in Figure 1(b). Observe that when the document structure changes, the position of all `player` nodes change but the locality of the `team` node still contains these two `player` nodes.

3.2 Defining Node Locality

Based on the aforementioned discussion, it is evident that a key issue associated with node locality is the identification of local nodes for a context node. In [15], we introduced the notion of locality as follows. A node n whose $\mathbb{T}(n) = t_n$ is *local* to the context node c whose $\mathbb{T}(c) = t_c$ if, among all pairs of nodes with type t_n and type t_c respectively, the *distance*[2] of n to c is minimum. That is, n is local to c *iff* $Distance(n, c) = min\{Distance(n', c')|c', n' \in \mathcal{N}, \mathbb{T}(n') = \mathbb{T}(n), \mathbb{T}(c') = \mathbb{T}(c)\}$.

[1] In this paper, we assume that the original and modified documents must have *same* content, *same* element labels, and real-world semantic relationships are maintained in both documents.

[2] The *distance* between nodes u and c is the number of edges in the unique, simple undirected path between u and c.

Note that based on this definition we can identify all the local nodes of the team node in Figures 1(a) and 1(b).

Although the aforementioned definition of local nodes works for many cases, for certain scenario it may fail to identify the local nodes correctly. Let us illustrate this by modifying the documents in Figure 1. Assume that there exists a predecessor node with value *San Diego Clippers* as a fourth child of the second team node in Figure 1(a). Similarly, the predecessor node is added as the second child of the second team node in Figure 1(b). Let us now consider the context node to be the championship node. Regardless of the structure of the XML document, the locality of a championship should include the team, the managers, the players, the league and the predecessor. Observe that the aforementioned definition of node locality now identifies the predecessor node (*San Diego Clippers*) as one of its local node. Semantically, *San Diego Clippers* is the predecessor node of *Los Angeles Clippers* and is not related to the championship node of *New York Knicks*. Hence, the locality of Knicks' championship node should exclude this predecessor node.

The reason the locality definition of [15] fails is because both championship and predecessor are *optional* nodes in this example. In fact, there is *only one* championship node and *only one* predecessor node. Therefore, they are clearly at the minimum distance of the pair of their types. Consequently, the predecessor node is always local to the championship node. In the following, we present a novel definition of node locality that addresses this limitation.

We first introduce some terminology to facilitate our exposition. For each type $t \in \mathcal{S}$ where $\mathcal{S}$ is the set of all types in the XML document $\mathbb{D}$, the *sub-type set* of t, denoted as S_t, is a subset of $\mathcal{S}$ including the types of all child nodes of all nodes with type t in $\mathbb{D}$. That is, $S_t = \{t' \in \mathcal{S} | \exists n, n' \in \mathcal{N}, n = Parent(n'), \mathbb{T}(n) = t, \mathbb{T}(n') = t'\}$. For example, considering the type team. There are two teams in the modified version of Figure 1(a). The child nodes of the first team (*New York Knicks*) are of types name, player, manager, championship and @founded. The child nodes of second one (*Los Angeles Clippers*) are of types name, player, manager, predecessor and @founded. Therefore, $S_{team} = \{name, player, manager, championship, predecessor, @founded\}$. Note that S_t can be computed while parsing the XML document if schema is not available. Otherwise, it can be computed from its schema/DTD.

In an XML document, a node n whose type is t is called a *full node*, denoted as $FullNode(n)$, if for all types in S_t, n has at least one child of that type. That is, $\forall t' \in S_t, \exists n' \in \mathcal{N}, n = Parent(n') \wedge \mathbb{T}(n') = t'$. If we denote the set of all types of all child nodes of n as T_n, then the above definition is equivalent to: $FullNode(n) = \texttt{true} \iff S_t \subseteq T_n$. Moreover, it is obvious that $T_n \subseteq S_t$. Therefore, $FullNode(n) = \texttt{true} \iff S_t = T_n$. For example, since S_{player} consists of only player's name and all nodes of type player in the modified document of Figure 1(a) has a child node with type player's name, all of them are full nodes. On the other hand, both team nodes are not full.

An XML document $\mathbb{D}$ is considered *full* (denoted by $Full(\mathbb{D})$) *iff* all of its nodes are full. That is, $\forall n \in \mathcal{N}$, $FullNode(n) = \texttt{true}$. Clearly in a full document, there are no optional nodes. Consequently, in a full document we can use minimum distance to

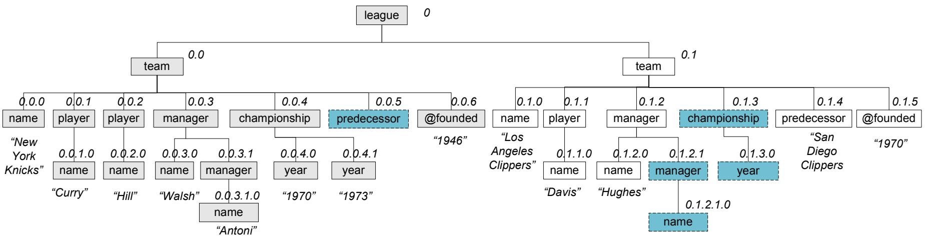


Fig. 2. Full document of `predecessor`-enhanced version of Figure 1(a)

identify node locality. That is, a node n whose $\mathbb{T}(n) = t$ is local to the context node c if, among all nodes with type t, the distance of n to c is minimum.

Definition 1. ***[Node Locality]*** *The locality of a node c in an* XML *document $\mathbb{D}$, denoted as $Locality(c)$, is a set of nodes in the full document $Full(\mathbb{D})$ in which each node n satisfies the following conditions: (a) n is in $Full(\mathbb{D})$; (b) $\forall n'$ in $Full(\mathbb{D})$, $\mathbb{T}(n') = \mathbb{T}(n) \implies Distance(c, n') \geq Distance(c, n)$.*

Next, we shall discuss how to convert any XML document $\mathbb{D}$ to a corresponding full document $Full(\mathbb{D})$. It is achieved by adding *ghost nodes* to $\mathbb{D}$. For each node n with type t and each type $t' \in S_t$ that n has no child nodes of type t', a new node n' with type t' is conceptually added to $\mathbb{D}$ as a child node of n. n' is called a *ghost node* since it does not actually exists in $\mathbb{D}$. For example, in the modified version of Figure 1(a), a `predecessor` node (ghost node) is added as the child of the first `team` node and a `championship` node is added as the child of the second `team` (*Los Angeles Clippers*). Notably, since $S_{championship}$ has type `year`, the `championship` node also has a ghost node `year`. Note that only one `year` node is sufficient to make the `championship` node full. Similarly, an assistant `manager` node (with accompanying `name` node) is added as the child of the second `manager` node (*Hughes*). Figure 2 depicts the full document (ghost nodes are shown in dotted blue rectangles). Note that no value nodes are added in the transformation as a full document does not require value nodes.

We can now compute locality of a node correctly using Definition 1. For instance, the `predecessor` node is no longer optional. Therefore, the `predecessor` node (*San Diego Clippers*) is now excluded from the locality of Knicks' `championship` node. Instead, the `predecessor` node with minimum distance to the context node is now the corresponding ghost node, which can be filtered out in the final results.

Definition 1 offers a straightforward method to compute the locality of a node c in document $\mathbb{D}$ by converting $\mathbb{D}$ to $Full(\mathbb{D})$ and finding the nodes in $Full(\mathbb{D})$ with minimal distance to c. However, this naive method has two drawbacks. First, $Full(\mathbb{D})$ is less space-efficient than $\mathbb{D}$ and may require updates every time $\mathbb{D}$ is updated. Second, locating the nodes with minimal distance to c requires traversal all nodes in $Full(\mathbb{D})$ at least once which is expensive when $Full(\mathbb{D})$ is large. In Section 5, we shall address these drawbacks by adopting an efficient strategy to evaluate the locality of c without transforming $\mathbb{D}$ to $Full(\mathbb{D})$ and with minimal node traversal.

4 Neighborhood Axis

The `neighborhood` axis is used to select *common* nodes of two context nodes. Informally, a node that is common to two nodes is semantically related to these nodes even when the document structure changes. Recall that the locality of a node is the set of all related (local) nodes to the context node. Therefore, the common nodes selected by the `neighborhood` axis should be in the locality of *both* input nodes. Observe that since node locality is structure-independent, the common locality of the two context nodes are identical even when structure of the document changes. For example, if John is interested in the common team of *Hill* and *Antoni*, the result should be the only `team` node in the common locality (team *New York Knicks*). On the other hand, if John asks for common `name` nodes associated to *Hill* and *Antoni*, then this query is ambiguous. In this case, the `neighborhood` axis should return all `name` nodes in the common locality of these two nodes.

Definition 2. *Let c_1 and c_2 be two context nodes and ℓ be the name test of the step. The **neighborhood** nodes of c_1 and c_2 with label ℓ, denoted as $neighborhood(c_1, c_2, \ell)$, is a list of nodes $[n_1, n_2, \ldots, n_j]$ where:*

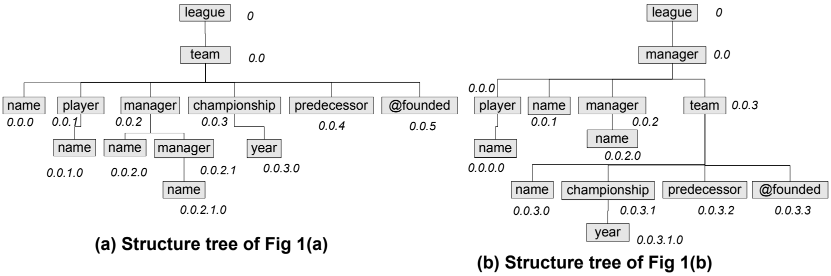
- $n_1, n_2, \ldots, n_j \in N$ *and* $j \geq 1$
- $\forall n_i \in neighborhood(c_1, c_2, \ell), \mathbb{L}(n_i) = \ell$
- $\forall n_i \in neighborhood(c_1, c_2, \ell), n_i \in Locality(c_1) \wedge n_i \in Locality(c_2)$
- $\forall p, q, 1 \leq p < q \leq j$, n_p *precedes* n_q *in document order*

The syntax for expressing `neighborhood` axis should consist of two input nodes (context nodes). One of them is the context node specified by the previous step. We refer to it as *left* context node. The other input node is a parameter (can be expressed as path expression), which we refer to as *right* context node. Hence, the BNF rules for `neighborhood` axis is as follow. First, the `neighborhood` is added into "$NonDirectionalStep$"[3]. Next, additional rules are specified to describe the `neighborhood` axis.

$$\begin{aligned} NonDirectionalStep &::= ClosestStep | RankDistanceStep | NeighborhoodStep \\ NeighborhoodStep &::= NeighborhoodAxis \quad NodeTest \\ NeighborhoodAxis &::= \texttt{"neighborhood""}\{\texttt{"}PathExpression\texttt{"}\}\texttt{"} \quad \texttt{"::"} \end{aligned}$$

Reconsider Figure 1 to find the common team of players *Hill* and *Curry*. Then, this query can be formulated as follows: Q_5: `//player [name='Hill']/ neighborhood{//player[name='Curry']}::team`. Observe that the query consists of three parts: (a) `//player[name='Hill']` is used to select the player *Hill* (left context node). (b) `//player[name='Curry']` is used to select the player *Curry* (right context node). (c) A `neighborhood` step with name test of `team` is used to find the common team of the two players. Observe that for both documents in Figure 1, Q_5 will return the `team` node whose name is "*New York Knicks*".

[3] We have introduced two additional non-directional axes, namely `closest` and `rank-distance`, in [15] and [1], respectively.



(a) Structure tree of Fig 1(a)

(b) Structure tree of Fig 1(b)

Fig. 3. Structure trees of modified versions of the XML documents in Figure 1

The `neighborhood` axis can also be used with a predicate. Suppose John now wishes to find the teammates of player *Hill*. Intuitively, a teammate is a player in the same team. Therefore, the XPath for this query can be expressed as follows: Q_6: `//player[neighborhood{//player[name='Hill']}::team]`.This query will return the `player` node whose name is "*Curry*" in both Figures 1(a) and 1(b).

5 Evaluation of Neighborhood Axis

In this section, we present a generic algorithm called SINBAD for evaluating neighborhood axis by exploiting node locality information. We begin by briefly introducing the terminology that we shall subsequently to describe the algorithm.

We denote the root-to-node path of a node n in an XML tree as $Path(n)$. That is, $Path(n)$ is a concatenation of the labels on the path from the root to n. Observe that $Path(n)$ is equivalent to the type of n ($\mathbb{T}(n)$). In the sequel, we shall use these two concepts interchangeably. Next we define the notion of *structure tree*. Given an XML document $\mathbb{D}$, the *structure tree* of $\mathbb{D}$, denoted as $\mathbb{S}_D$, is a DataGuide structural summary [5] representing all unique paths in $\mathbb{D}$. That is, each unique path p in $\mathbb{D}$ is represented in $\mathbb{S}_D$ by a node whose path from the root node to this node is p. Further, every unique label path of $\mathbb{D}$ is described exactly once, regardless of the number of times it appears in $\mathbb{D}$. The structure tree encodes no path that does not appear in $\mathbb{D}$. Note that a structure tree can be computed in linear time for tree-structured data [5]. Figure 3 depicts the structure trees of the modified `predecessor`-enhanced documents in Figure 1 (the nodes are encoded with their Dewey labels). Intuitively, a document $\mathbb{D}$ and its full document version $Full(\mathbb{D})$ share a common structure tree.

Lemma 1. *Given a document* $\mathbb{D}$*, the structure trees of* $\mathbb{D}$ *and* $Full(\mathbb{D})$ *are identical.*

Proof. (Sketch) According to the definition of full documents, for any type $t \in \mathcal{S}$, its subtype set is identical in $\mathbb{D}$ and in $Full(\mathbb{D})$. In the structure tree, each type t corresponds to a path p and each subtype of t corresponds to a child of p. Thus, in both $\mathbb{D}$ and $Full(\mathbb{D})$, the children list of all nodes in the structure tree are identical. Hence, their structure trees are identical. ∎

For example, Figure 3(a) depicts the structure tree of both the document in Figure 1(a) and its full document in Figure 2.

Given two paths p_1 and p_2 in $\mathbb{S}_D$, the *path distance* between p_1 and p_2, denoted as $Distance(p_1, p_2)$, is the length of path connecting the nodes represented by p_1 and p_2. The *level* of p_1, denoted as $Level(p_1)$, is the length of p_1. The LCA of p_1 and p_2, denoted as $LCA(p_1, p_2)$, is the longest common prefix of p_1 and p_2. Finally, the LCA *level* of p_1 and p_2, denoted as $LCALevel(p_1, p_2)$, is the level of the $LCA(p_1, p_2)$. That is, $LCALevel(p_1, p_2) = Level(LCA(p_1, p_2))$.

5.1 Evaluation of Locality

In this section, we present a set of properties that shall be exploited by Algorithm SINBAD (Section 5.2) to efficiently check whether a node is in the locality of a context node.

Lemma 2. *Let n_1 and n_2 be two nodes in a document $\mathbb{D}$ and $p_1 = Path(n_1)$ and $p_2 = Path(n_2)$ in structure tree $\mathbb{S}_D$. Then,*

$$Distance(n_1, n_2) = Level(n_1) + Level(n_2) - 2 \times LCALevel(n_1, n_2)$$
$$Distance(p_1, p_2) = Level(p_1) + Level(p_2) - 2 \times LCALevel(p_1, p_2)$$

Proof. (Sketch) The path connecting two nodes n_1 and n_2 in a tree is unique and this path must pass through the $LCA(n_1, n_2)$. Therefore, we can divide this path into two parts: one from n_1 to $LCA(n_1, n_2)$ and another from $LCA(n_1, n_2)$ to n_2. The length of the path from n_1 to $LCA(n_1, n_2)$ is equal to $Level(n_1) - LCALevel(n_1, n_2)$ while the length of the path from $LCA(n_1, n_2)$ to n_2 is $Level(n_2) - LCALevel(n_1, n_2)$. Hence, $Distance(n_1, n_2)$ is equal to the sum of these two subparts and is equal to $Level(n_1)$ + $Level(n_2) - 2 \times LCALevel(n_1, n_2)$. The proof is similar for $Distance(p_1, p_2)$. ∎

Theorem 1. *Let n_1 and n_2 be two nodes in a document $\mathbb{D}$. Let $p_1 = Path(n_1)$ and $p_2 = Path(n_2)$ be two paths in $\mathbb{S}_D$. Then, (i) $LCALevel(n_1, n_2) \leq LCALevel(p_1, p_2)$ and (ii) $Distance(n_1, n_2) \geq Distance(p_1, p_2)$.*

Proof. (Sketch) From Lemma 2, it is clear that (i) and (ii) are equivalent (notice that $Level(n) = Level(path(n)) \forall n$). Thus, we only need to prove (i).

Let n be $LCA(n_1, n_2)$ and $p = Path(n)$. Since n is an ancestor of both n_1 and n_2, p is a prefix of both p_1 and p_2. Therefore, p is an ancestor of both p_1 and p_2. Based on the definition of LCA level, we have: $Level(p) \leq LCALevel(p_1, p_2)$. Therefore: $LCALevel(n_1, n_2) = Level(n) = Level(p) \leq LCALevel(p_1, p_2)$. ∎

For example, in Figure 2, let n_1 and n_2 be nodes whose Dewey codes are `0.0.1` and `0.1.4`, respectively. Then $LCALevel(n_1, n_2) = 1$ and $Distance(n_1, n_2) = 4$. In Figure 3(a), the paths of n_1 and n_2 (p_1, p_2) are nodes whose Dewey codes are `0.0.1` and `0.0.4`, respectively. Their LCA level is 2 and their distance is 2. Hence, $LCALevel(n_1, n_2) < LCALevel(p_1, p_2)$ and $Distance(n_1, n_2) > Distance(p_1, p_2)$.

Theorem 2. *Let c be a node in the full document $Full(\mathbb{D})$ ($Path(c) = p_c$). Then, for any path p_n in $\mathbb{S}_D$, there exists a node $n \in Full(\mathbb{D})$ whose path is p_n such that (i) $LCALevel(c, n) = LCALevel(p_c, p_n)$ and (ii) $Distance(c, n) = Distance(p_c, p_n)$.*

Proof. (Sketch) Following Lemma 2, (i) and (ii) are equivalent. Thus, we only need to prove (i). Let $\ell = LCALevel(p_c, p_n)$. Note that $\ell \leq Level(p_c) = Level(c)$. Let a_ℓ be the ancestor (or self) of c at level ℓ. Choose $a_{\ell+1}$ as a child of a_ℓ whose tag is equal to the tag of p_n at level $\ell + 1$ of the structure tree $\mathbb{S}_D$. Since $\ell = LCALevel(p_c, p_n)$, this tag is not in p_c and therefore, is not the tag of the ancestor of c at level $\ell + 1$. Therefore, the chosen $a_{\ell+1}$ is not the ancestor of c at level $\ell + 1$.

Similarly, $a_{\ell+2}$ is selected as a child of $a_{\ell+1}$ whose tag is the tag of p_n at level $\ell + 2$. Continue this process repeatedly until $k = Level(p_n)$ where a_k is the node n that we need to find. Obviously, $Path(a_k) = p_n$. Moreover, since a_ℓ is a common ancestor at level ℓ of c and a_k and their ancestors at level $\ell+1$ are different (since they have different tags), $a_\ell = LCA(c, a_k)$. Hence, $\ell = LCALevel(c, a_k)$. So, $LCALevel(p_c, p_n) = \ell = LCALevel(c, a_k)$. ∎

For example, consider the full document in Figure 2 and the corresponding structure tree in Figure 3(a). Let c be the node whose Dewey code is `0.0.3`. Hence, p_c is `league.team.manager` which has a Dewey code of `0.0.2` in the structure tree. Let's choose p_n as `league.team.predecessor` (Dewey code is `0.0.4` in Figure 3(a)). In the structure tree, $LCALevel(p_c, p_n) = 2$. Then $p_2 = LCA(p_c, p_n)$ is the path `league.team` representing the type `team`. Observe that the level 2 ancestor, a_2, of c is the node `0.0` (clearly, the path of a_2 is p_2).

The tag at level 3 of p_n is `predecessor`. Let p_3 be the level 3 ancestor of p_n. Then p_3 is `league.team.predecessor`. Since p_3 is a child of p_2 in the structure tree, there exists nodes which have path p_3 and are the child nodes of nodes with path p_2 in the full document. Hence, type `predecessor` is a type in the sub-type set of type `team`. Consequently, a_2 must have a child node with type `predecessor` (path p_3). Let a_3 be this child node of a_2. Then a_3 is the node n we need to find. Observe that a_3 is a ghost node in Figure 2 with Dewey code of `0.0.5` and $LCALevel(c, a_3) = LCALevel(p_c, p_n) = 2$ and $Distance(c, a_3) = Distance(p_c, p_n) = 2$.

Given a context node c whose path is p_c and a set of nodes N in a full document $Full(\mathbb{D})$ whose path is p_n, Theorem 1 shows that $Distance(p_c, p_n)$ is a lower bound of the distance between c and any nodes $n \in N$. On the other hand, Theorem 2 shows that $\exists n \in N, Distance(c, n) = Distance(p_c, p_n)$. Thus, $Distance(p_c, p_n)$ is the minimum distance between c and any nodes in N. Following Definition 1, all nodes $n \in Locality(c)$ must satisfy $Distance(c, n) = Distance(p_c, p_n)$.

Theorem 3. *Let c be the context node in document $\mathbb{D}$. Then a node $n \in Locality(c)$ iff (a) $Distance(c, n) = Distance(p_c, p_n)$ or (b) $LCALevel(c, n) = LCALevel(p_c, p_n)$.*

Proof. (Sketch) Condition (a) is a direct consequence of Definition 1, Theorem 1 and Theorem 2. Condition (b) can be proved by exploiting Lemma 2. Since $Distance(c, n) = Distance(p_c, p_n)$, $Level(c) + Level(n) - 2LCALevel(c, n) = Level(p_c) + Level(p_n) - 2LCALevel(p_c, p_n)$. Hence, $LCALevel(c, n) = LCALevel(p_c, p_n)$. ∎

For example, reconsidering Figure 2. Let c be the node whose Dewey code is `0.0.3` (the head `manager` node). Let us find the `player` nodes that are local to c. Here p_c = `league.team.manager` and p_n = `league.team.player`. Therefore, $LCALevel(p_c, p_n) = 2$. Observe that in Figure 2, there are three nodes having path p_n.

Algorithm 1: The Algorithm SINBAD.

```
Input: A document $\mathbb{D}$, context nodes $c_1$ and $c_2$ in $\mathbb{D}$, name test $\ell$, set $P$ of all paths in $\mathbb{S}_D$
Output: A set of neighborhood nodes $Results$
Initialize $NeighborhoodPaths = \infty$ ;
Initialize $Results = \infty$ ;
for (each $p \in P$) do
    if ($LastTag(p) == \ell$) then
        Add $p$ into $NeighborhoodPaths$;
for $p \in NeighborhoodPaths$ do
    $a_1 \leftarrow$ the ancestor (or self) of $c_1$ at level $LCALevel(Path(c_1), p)$;
    $a_2 \leftarrow$ the ancestor (or self) of $c_2$ at level $LCALevel(Path(c_2), p)$;
    if $a_1$ is an ancestor-or-self of $a_2$ or $a_2$ is ancestor-or-self of $a_1$ then
        $a \leftarrow$ the descendant between $a_1$ and $a_2$;
        Add all descendants-or-self of $a$ in $\mathbb{D}$ whose path is $p$ into $Results$;
return $Results$
```

Their Dewey codes are 0.0.1, 0.0.2, and 0.1.1. Their $LCALevel$ with c are 2, 2, and 1, respectively. Hence, according to Theorem 3 only nodes 0.0.1 and 0.0.2 are in the locality of c. Observe that these two nodes represent the players *Curry* and *Hill* who are managed by the head manager *Walsh*.

Remark. Theorem 3 offers a very efficient technique to evaluate locality due to three reasons. First, $LCALevel(p_c, p_n)$ can be computed completely from the structure tree $\mathbb{S}_D$ whose size is significantly smaller than $\mathbb{D}$ in most practical cases. $\mathbb{S}_D$ can also be built directly from $\mathbb{D}$ without creating $Full(\mathbb{D})$ (Lemma 1). Second, due to Theorem 1, $LCALevel(c, n) = LCALevel(p_c, p_n) = \ell$ is equivalent with n is a descendant (or self) of the ancestor-or-self node a of c at level ℓ. Notice that if our list of nodes are sorted in document order, the descendant list of a are consecutive and usually small so that traversing them is usually cheap. Third, observe that users are not interested in ghost nodes. Hence, these nodes need to be filtered out in the output. Theorem 3 allows us to accomplish this efficiently *without transforming the original document* $\mathbb{D}$ *to a full document*. We only traverse the descendant list of a in the original document $\mathbb{D}$ so that all result nodes are actual nodes. If a local node n (*i.e.,* descendant of a) in the full document is not returned, it means that n is a ghost node.

For example, let c be the node with Dewey code 0.0 (team node). We wish to find the predecessor nodes that are local to c. Since p_c = league.team and p_n = league.team.predecessor, $LCALevel(p_c, p_n) = 2$. The ancestor-or-self of c at level 2 is c itself. In the full document there are two nodes with type p_n (0.0.5 and 0.1.4). However, since we only need to traverse the descendant list of node 0.0 in the original document $\mathbb{D}$, neither would be traversed and there does not exist any predecessor node that is in the locality of c in the original document (Fig.1). Notably, although node 0.0.5 is descendant of node 0.0 and local to c in the full document, since we traverse only 0.0's descendant list in the original document (we do not transform it to a full document), node 0.0.5 would not be returned.

5.2 Algorithm SINBAD

Algorithm 1 outlines the SINBAD algorithm. Most importantly, Algorithm 1 does not take $Full(\mathbb{D})$ as input or require $Full(\mathbb{D})$ in any of its steps. We illustrate the steps with an example. Reconsider the query Q_5 which selects the common team of player *Hill* and manager *Antoni*. The two context nodes in this query are the `player` node (`0.0.2`) and the `manager` node (`0.0.3.1`). Lines 3-5 are used to find all paths whose last tag is the label `team`. In our example, the only such path is `league.team`. For each path $p \in NeighborhoodPaths$, Lines 7-8 are used to find the ancestor of c_1 and c_2 at level $LCALevel(p, path(c_1))$ and $LCALevel(p, path(c_2))$, respectively. According to Theorem 3, all result nodes must be descendants of *both* ancestor nodes a_1 and a_2. Hence, a_1 and a_2 must have ancestor-descendant relation (Line 9). Let a be the descendant node between a_1 and a_2 (Line 10), all results must be descendants (or self) of a (Line 11). Notice that Line 11 finds the descendants of a in $\mathbb{D}$, not $Full(\mathbb{D})$, due to reasons mentioned in Section 5.1. Therefore, Algorithm 1 does *not* require $Full(\mathbb{D})$ or convert $\mathbb{D}$ to $Full(\mathbb{D})$. Specifically, for our example, both a_1 and a_2 would be node `0.0` and the only descendant-or-self nodes with label `team` and descendant of `0.0` are that node itself. It is our only result.

Time Complexity. Let k be the number of paths satisfying the neighborhood path condition (Lines 3-5) and the $Desc(p)$ be the set of descendant-or-self node of node a produced in Lines 10-11. Assume that $LCALevel()$ and ancestor-descendant evaluation could be computed in $O(1)$ time. The time complexity of the algorithm is: $O(|P| + 3k + \sum_{p \in P} |Desc(p)|)$. Notice that both $|P|$ and k are usually very small so that the worst-case complexity is usually dominated by $\sum_{p \in P} |Desc(p)|$. Furthermore, since all nodes in $Desc(p)$ have path p, all nodes in $Results$ are unique and $\sum_{p \in P} |Desc(p)|$ happens to be our result size which is the expected lower bound of our algorithm. Moreover, we can also notice that the input context nodes are only used for LCA computation and ancestor-descendant checking, both can be achieved using only the node identifiers (*e.g.,* Dewey code). Hence, retrieving data for a context node is cheaper than retrieving data for a result node.

6 Performance Study

We present the experiments conducted to evaluate the performance of our proposed axis and report some of the results obtained. Note that SINBAD is independent of any specific storage scheme for XML data. In this paper, we have realized it on top of a *path materialization*-based [6] relational storage scheme called SUCXENT++ [10] in Java JDK1.6. Due to space constraints, we do not discuss the implementation of SQL translation strategy for SINBAD. Note that our implementation *does not* require any invasion of the database kernel. All of our experiments are conducted on an Intel Core 2 Quad CPU 2.66GHz machine running on Windows XP Service Pack 3 with 2GB RAM. The RDBMS used was MS SQL Server 2008 Developer Edition.

Id	Size	No. of Attributes	No. of Leaf Elements	Max Depth
DC10	11MB	15,000	152,673	8
DC100	111MB	150,000	1,520,322	8
DC1000	1111MB	1,500,000	15,215,868	8

(a) XBENCH data sets

Id	Size	No. of Attributes	No. of Leaf Elements	Max Depth
U28	28MB	771,877	422,972	6
U284	284MB	7,791,617	4,230,003	6
U2843	2843MB	77,977,270	42,421,745	6

(b) UniProt data sets

Fig. 4. Datasets

Id	Query	No. of right context nodes			No. of result nodes		
		DC10	DC100	DC1000	DC10	DC100	DC1000
XQ1	//subject/neighborhood{//title}::ISBN	2500	25000	250000	2500	25000	250000
XQ1'	//item[subject][title]//ISBN						
XQ2	//subject/neighborhood{//title}::name	2500	25000	250000	23385	232290	2375909
XQ2'	//item[subject][title]//name						
XQ3	//subject/neighborhood{//author/date_of_birth}::ISBN	6295	62420	625303	2500	25000	250000
XQ3'	//item[subject][//author/date_of_birth]//ISBN						
XQ4	closest::subject/neighborhood{closest::phone_number}::ISBN	2500	25000	250000	2500	25000	250000
XQ4'	//item[subject][publisher//phone_number]//ISBN						

(a) XBench query sets

Id	Query	No. of right context nodes			No. of result nodes		
		U28	U284	U2843	DC10	DC100	DC1000
UQ1	//protein/neighborhood{//feature}::evidence	150255	1382599	13820779	9618	74778	725820
UQ1'	//entry[//protein][//feature]//evidence						
UQ2	//protein/neighborhood{//feature}::reference	150255	1382599	13820779	411665	4084917	40821754
UQ2'	//entry[//protein][//feature]//reference						
UQ3	//protein/neighborhood{//entry/dbReference}::evidence	347754	3601383	36053637	9510	74136	720072
UQ3'	//entry[//protein][dbReference]//evidence						
UQ4	closest::protein/neighborhood{closest::gene}::evidence	13000	137292	1378484	9024	69924	678378
UQ4'	//entry[//protein][gene]//evidence						

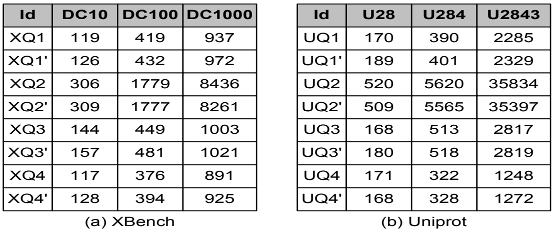
(b) UniProt query sets

Fig. 5. Querysets

Data and Query Sets. We use XBench DCSD [14] as a synthetic data set and Uniprot/KB XML[4] as a real-world data set. We vary the size of XML documents from 10MB to 1GB for XBench DCSD data set. Since the original UNIPROT data is 2.8GB in size (denoted as U2843), we also truncated this document into smaller XML documents of sizes 28MB and 284MB (denoted as U28 and U284, respectively) to study scalability. Figure 4 depicts the characteristics of the data sets. Figure 5 depicts the query sets for XBench DCSD (XQ1–XQ4) and Uniprot/KB data sets (UQ1–UQ4). Note that queries XQ4 and UQ4 showcase usage of `neighborhood` axis with other non-directional axis (*i.e.*, `closest` [15]). We also consider equivalent directional XPath queries (same results set) of XQ1–XQ4 (denoted as XQ1$'$ - XQ4$'$) and UQ1–UQ4 (UQ1$'$ - UQ4$'$) in order to compare the performance of structure-independent queries with their directional counterparts.

Test Methodology. Appropriate indexes were constructed for SUCXENT++. Prior to our experiments, we ensured that statistics had been collected. The bufferpool of the

[4] Downloaded from `ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/uniprot_sprot.xml.gz`



Id	DC10	DC100	DC1000
XQ1	119	419	937
XQ1'	126	432	972
XQ2	306	1779	8436
XQ2'	309	1777	8261
XQ3	144	449	1003
XQ3'	157	481	1021
XQ4	117	376	891
XQ4'	128	394	925

(a) XBench

Id	U28	U284	U2843
UQ1	170	390	2285
UQ1'	189	401	2329
UQ2	520	5620	35834
UQ2'	509	5565	35397
UQ3	168	513	2817
UQ3'	180	518	2819
UQ4	171	322	1248
UQ4'	168	328	1272

(b) Uniprot

Fig. 6. Performance results (in msec.)

RDBMS was cleared before each run. Each query was executed six times and the results from the first run were always discarded.

Experimental Results. Figure 6 reports the performance neighborhood axis queries. We can make the following observations. Firstly, *the execution time increases sub-linearly with result size*. Notice that XQ1 and UQ1 have identical context node set with XQ2 and UQ2, respectively, but with different result sizes. In particular, the result size of XQ2 is nearly 10 times larger than that of XQ1 and the result size of UQ2 is about 50 times larger than that of UQ1. However, the execution times grows at much slower rate compared to the result size. Secondly, *the execution time increases sub-linearly with the number of right context nodes* (recall from Section 4). For instance, although the numbers of right context nodes of XQ3 and UQ3 are significantly larger than XQ1 and UQ1, respectively, there is not much difference in the corresponding execution times. This is consistent with our discussion in Section 5.2. Thirdly, *our proposed technique is scalable* as the execution time increases linearly to the document size. Notice that for 75% of queries, the execution time on even the largest dataset is less than 3 seconds. Lastly, there are *no significant performance difference between the neighborhood axis queries and their corresponding directional counterparts* (XQ1′ - XQ4′ and UQ1′ - UQ4′). This highlights the strength of our approach as users can query in a structure-independent manner without compromising on query performance.

7 Conclusions and Future Work

The quest for structure-independent querying of XML data has become more pressing due to inability of end-users to be aware of structural details of underlying data. In this paper, we present a novel structure-independent, non-directional XPath axis, called the neighborhood axis, to locate common neighbors of two context nodes. We proposed an algorithm called SINBAD that exploits the notion of node locality and small size of XML structural summary to efficiently abstract the common nodes of a pair of context nodes. Our empirical study on top of an existing path materialization-based relational storage showed that SINBAD has excellent real-world performance. In future, we plan to extend our approach to yet other non-directional axes, which we believe can be supported using the techniques presented in this paper.

References

1. Bhowmick, S.S., Dyreson, C., et al.: Towards Non-Directional XPath Evaluation in a RDBMS. In: CIKM (2009)
2. Brodianskiy, T., Cohen, S.: Self-Correcting Queries in XML. In: CIKM (2007)
3. Cohen, S., Mamou, J., Kanza, Y., Sagiv, Y.: XSEarch: A Semantic Search Engine for XML. In: VLDB (2003)
4. Curino, C.A., Moon, H.J., Zaniolo, C.: Graceful Database Schema Evolution: The Prism Workbench. In: VLDB (2008)
5. Goldman, R., Widom, J.: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In: VLDB (1997)
6. Gou, G., Chirkova, R.: Efficiently Querying Large XML Data Repositories: A Survey. IEEE TKDE 19(10) (2007)
7. Li, Y., Yu, C., Jagadish, H.V.: Schema-Free XQuery. In: VLDB (2004)
8. Liu, Z., Chen, Y.: Identifying Meaningful Return Information for XML Keyword Search. In: SIGMOD (2007)
9. Liu, Z., Chen, Y.: Reasoning and Identifying Relevant Matches for XML Keyword Search. In: VLDB (2007)
10. Soh, K.H., Bhowmick, S.S.: Efficient Evaluation of not-Twig Queries in A Tree-Unaware RDBMS. In: Yu, J.X., Kim, M.H., Unland, R. (eds.) DASFAA 2011, Part I. LNCS, vol. 6587, pp. 511–527. Springer, Heidelberg (2011)
11. Termehchy, A., Winslett, M., Chodpathumwan, Y.: How Schema Independent Are Schema Free Query Interfaces? In: ICDE (2011)
12. Velegrakis, Y., Miller, R.J., Popa, L.: Mapping Adaptation Under Evolving Schemas. In: VLDB (2003)
13. Xu, Y., Papakonstantinou, Y.: Efficient Keyword Search for Smallest LCAs in XML Databases. In: SIGMOD (2005)
14. Yao, B., Tamer Özsu, M., Khandelwal, N.: XBench: Benchmark and Performance Testing of XML DBMSs. In: ICDE (2004)
15. Zhang, S., Dyreson, C.: Symmetrically Exploiting XML. In: WWW (2006)

Top-Down SLCA Computation Based on List Partition

Junfeng Zhou[1], Zhifeng Bao[2], Ziyang Chen[1],
Guoxiang Lan[1], Xudong Lin[1], and Tok Wang Ling[2]

[1] School of Information Science and Engineering, Yanshan University
[2] School of Computing, National University of Singapore
{zhoujf,zychen,gxlan,xdlin}@ysu.edu.cn,
{baozhife,lingtw}@comp.nus.edu.sg

Abstract. In this paper, we focus on efficient processing of a given XML keyword query based on SLCA semantics. We propose an efficient algorithm that processes all nodes in the set of inverted Dewey label lists in a *top-down* way. Specifically, our method *recursively* divides the set of initial Dewey label lists into a set of *minimum nontrivial block*s (*MNBlock*s), where a block consists of a set of Dewey label lists and corresponds to an XML tree. The "minimum" means that for a given block, none of its sub-blocks corresponds to a subtree that contains all keywords of the given query; the "nontrivial" means that no block can contain an empty list. Based on these MNBlocks, our method produces all qualified results by directly outputting the LCA node of all nodes in each MNBlock as a qualified SLCA node. During processing, our method can intelligently prune useless keyword nodes according to the distribution of all nodes in a given block. Our experimental results verify the performance advantages of our method according to various evaluation metrics.

1 Introduction

Keyword search over XML data has attracted a lot of research efforts [1–3,5–8,11,12] in the last decade, where a core problem is how to efficiently answer a given keyword query. Typically, an XML document can be modeled as a node-labeled tree T, and for a given keyword query Q, lowest common ancestor (LCA) is the basis of existing XML keyword search semantics [5,6,11,12], of which the most widely followed variant is smallest LCA (SLCA) [8,11]. Each SLCA node v of Q on T satisfies that v is an LCA node of Q on T, and no other LCA node of Q can be v's descendant node. The meaning of SLCA semantics is straightforward, i.e., smaller trees contain more meaningfully related nodes.

To facilitate SLCA computation on XML data, existing methods [8,11] assign to each node v a Dewey [9] label, and all Dewey labels of nodes that directly contain a certain keyword k_i are organized in an inverted list L_i in document order. The state-of-the-arts include the Stack [11], IL [11] and IMS [8] algorithms, and the main idea is utilizing the positional relationships of nodes in the inverted lists to make semantic pruning in a *bottom-up* way, that is, they process all nodes that directly contain keywords of the given query and check whether their LCA nodes satisfy the requirement of SLCA semantics.

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 172–184, 2012.
© Springer-Verlag Berlin Heidelberg 2012

The Stack algorithm sequentially processes all Dewey labels in document order by using a stack to merge Dewey labels on the fly and simultaneously computing qualified SLCA nodes. The IL algorithm computes the SLCA results by processing two lists each time from the shortest to the longest, i.e., $SLCA(Q) = SLCA(SLCA(L_1, ..., L_{m-1}), L_m)$. Instead of breaking the SLCA computation into a series of binary SLCA computations as IL does, the IMS algorithm computes each potential SLCA by taking one node from each Dewey label list in each iteration. For the above three algorithms, the Stack algorithm is the most straightforward in each iteration, but usually suffers from huge number of iterations. Compared with Stack, IL and IMS are more flexible to utilize the positional relationship to prune useless keyword nodes. The difference between IL and IMS lies in that in each iteration, IL *always* uses a node of the shorter list to probe the longer list, while IMS always uses a *maximum* node to probe other lists. IL is simpler than IMS in each iteration, while usually suffers from more iterations than IMS. IMS needs the least number of iterations, but suffers from the hightest cost in each iteration. Therefore, IMS could be the best choice when all nodes of the set of lists are not uniformly distributed; IL could perform best when all nodes are uniformly distributed and there exists huge difference in lengths of the set of lists; otherwise, Stack is the best choice.

In practice, different keyword queries may correspond to Dewey label lists with *different* distributions; even worse, for a certain keyword query $Q = \{k_1, k_2, ..., k_m\}$ and the set of Dewey label lists $\mathcal{L} = \{L_1, L_2, ..., L_m\}$, *different parts* of $\mathcal{L}$ may possess *different* distributions, which result in the inefficiency for existing methods, that is, according to only the *overall distribution* of all nodes and therefore fixing a certain method could result in losing chances of making fine-grained optimization.

Example 1. Consider processing $Q = \{a, b\}$ on D in Fig. 1. For all uniformly distributed nodes under x_4, the number of iterations of Stack, IL and IMS are 200, 100 and 99, respectively. Although Stack needs the most iterations, it is the most efficient since the cost of each iteration for Stack is much less than that of IL and IMS. Consider processing $Q = \{a, b\}$ on Case 3 and Case 4, where all nodes are not uniformly distributed, and both IL and IMS are much more efficient than Stack in such a case, since they can skip most useless nodes. If Case 3 and Case 4 are processed *separately*, IL will be more efficient than IMS, since both IL and IMS need the same number of iterations, but IMS suffers from higher cost than IL in each iteration. However, if Case 3 and Case 4 are processed *together*, that is, $L_a = \{a_{302}, ..., a_{401}\}$, $L_b = \{b_{302}, ..., b_{402}\}$, we have $|L_a| < |L_b|$, then IMS is much more efficient than IL, because in each iteration, IL always uses a node from L_a to probe L_b, which means that the probe operations of a_{302} to a_{398} are useless since they produce the same result as that of a_{399}.

To realize *fine-grained optimization* for SLCA computation, we propose an efficient algorithm that *recursively divides the set of initial Dewey label lists into a set of minimum nontrivial blocks (MNBlocks) in a top-down manner, then outputs the LCA node of all nodes in each MNBlock as an SLCA node*, where a block consists of a set of Dewey label lists and corresponds to an XML tree, the "minimum" means that for a given block, no one of its sub-block corresponds to a subtree that contains all keywords, the "nontrivial" means that every list of a block is a nonempty list. During processing, the

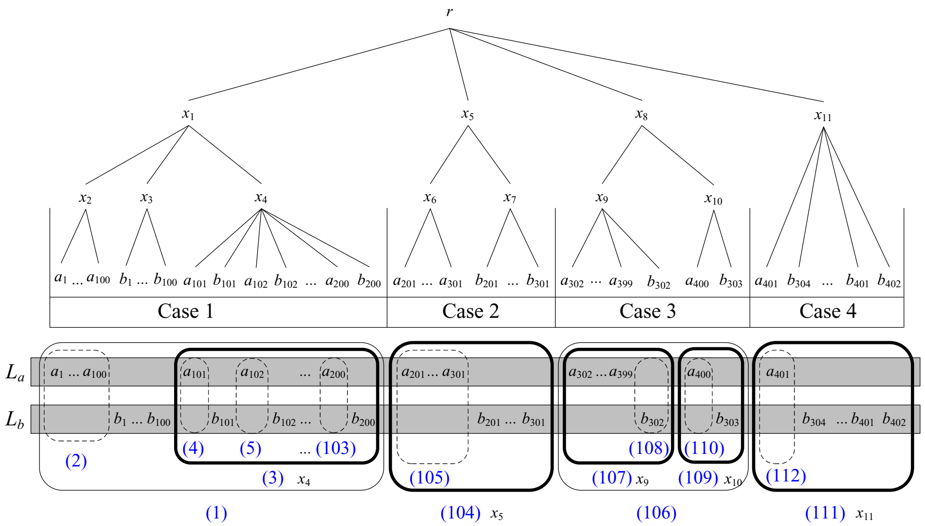


Fig. 1. A sample XML document D, where only nodes that directly or inderectly contain a and b are kept for explanation. L_a and L_b are the two inverted Dewey label lists for $Q = \{a, b\}$, where all nodes in the two lists are shown in document order; each solid (dashed) rectangle on L_a and L_b represents a nontrivial (trivial) block, which corresponds to a subtree and is marked by an enclosed number.

"fine-grained optimization" in our method amounts to an intelligent pruning of useless keyword nodes according to the distribution of all nodes in a given block.

Intuitively, our method takes all nodes in the set of inverted Dewey label lists as leaf nodes of an XML tree T, and checks whether it contains all keywords of the given query. The "top-down" processing strategy means that if T contains all keywords, T must contain at least one SLCA node, we then remove the root node of T and get a forest $\mathcal{F}_T = \{T_1, T_2, ..., T_n\}$ of subtrees corresponding to the set of child nodes of T's root node. Based on $\mathcal{F}_T$, we check whether each subtree contains all keywords. If no subtree in $\mathcal{F}_T$ contains all keywords, it means that T is a smallest tree that contains all keywords, then we directly output T's root node as an SLCA node; otherwise, for each subtree in $\mathcal{F}_T$ that contains all keywords, we just need to *recursively* compute its subtree set until no subtree in a subtree set contains all keywords. For a given tree T that contains all keywords of the given query, to check whether there exists some subtrees of T that contain all keywords, our method does not need to check every subtree in $\mathcal{F}_T$; instead, we just need to firstly find the keyword k_i with the smallest number of occurrences in T, then find all subtrees of T that contain k_i; at last, for each of these subtrees, we check whether it contains all other keywords. In this way, our method realizes the fine-grained optimization, thus can be adaptive to inverted Dewey label lists with different kinds of keyword distributions.

Example 2. Continue Example 1. Let T_v be the subtree rooted at node v, our method firstly finds block (1) corresponds to T_{x_1} in Fig. 1. Since block (1) contains both "a" and "b", our method recursively finds two blocks, i.e., block (2) and (3), corresponding to T_{x_2} and T_{x_4}, respectively. Note that all nodes under x_3 are not processed since our method uses nodes of L_a to find blocks. As block (2) does not contain any node of L_b (see rectangle marked by "(2)"), we know that subtree T_{x_2} does not contain all keywords, thus its root node, i.e., x_2, is not an SLCA node and we skip all nodes under x_2. As block (3) contains nodes of both L_a and L_b, we use nodes of L_a to find more sub-blocks, shown as block (4) to block (103), of which no one contains node of L_b, thus we know that the root node of block (3), i.e., x_4, is an SLCA node. The next block computed in our method is block (104) corresponding to T_{x_5}. Since T_{x_5} contains all keywords, we recursively find block (105). As block (105) is the only sub-block found in block (104) and it does not contain node of L_b, we know the root node of T_{x_5}, i.e., x_5, is an SLCA node. The following processing is similar, and the only thing that needs to be noted is that when processing block (106), our method uses nodes of L_b to compute sub-blocks, because for block (106), the number of nodes in L_b is much less than that in L_a. For block (111), we use nodes of L_a again to compute sub-blocks, since the number of nodes in L_a is much less than that in L_b for block (111). Finally, our method outputs the root nodes of T_{x_4}, T_{x_5}, T_{x_9}, $T_{x_{10}}$ and $T_{x_{11}}$ as qualified SLCA nodes, which correspond to block (3), (104), (107), (109) and (111), respectively.

The rest of the paper is organized as follows. In Section 2, we introduce preliminaries and related work. In Section 3, we introduce partition, block, MPartition, EBSet and MNBlock, and their important properties. Our LPSLCA algorithm is presented in Section 4. In Section 5, we present the experimental results, and conclude our paper in Section 6.

2 Preliminaries and Related Work

2.1 Data Model

We model an XML document as a node-labeled tree, where nodes represent elements or attributes, while edges represent direct nesting relationship between nodes in the tree. If a keyword k appears in the node name or attribute name, or k appears in the text value of v, we say v directly contains k. Fig. 1 is a sample XML document.

The positional relationships between two nodes include Document Order $(\prec_d)$, Equivalence (=), AD (ancestor-descendant, $\prec_a$), PC (parent-child, $\prec_p$), Ancestor-or-self $(\preceq_a)$ and Sibling relationship. $u \prec_d v$ means that u is located before v in document order, $u \prec_a v$ means that u is an ancestor node of v, $u \prec_p v$ denotes that u is the parent node of v. If u and v represent the same node, we have $u = v$, and both $u \preceq_d v$ and $u \preceq_a v$ hold. During processing, each node is assigned with a Dewey label to facilitate the computation of various positional relationships. In the following discussion, we do not differentiate between a node and its label if without ambiguity.

2.2 Query Semantics and Related Algorithms

For a given query $Q = \{k_1, k_2, ..., k_m\}$ and an XML document D, inverted Dewey label lists are often built to record which nodes directly contain which keywords. We use L_i to denote the inverted Dewey label list of k_i, of which all Dewey labels are sorted in document order. Let $S = \{v_1, v_2, ..., v_n\}$ be a set of nodes, $lca(S) = lca(v_1, v_2, ..., v_n)$ denotes the lowest common ancestor (LCA) of all nodes in S. For a set of node sets $\mathbb{S} = \{S_1, S_2, ..., S_m\}$, $lca(\mathbb{S})$ represents the LCA of all nodes in $\bigcup_1^m S_i$.

The LCAs of Q on D are defined as $LCA(Q) = LCA(L_1, L_2, ..., L_m) = \{v|v = lca(v_1, v_2, ..., v_m), v_i \in L_i(1 \leq i \leq m)\}$. E.g., the LCA nodes for $Q = \{a, b\}$ on D in Fig. 1 include r, x_1, x_4, x_5, x_8, x_9, x_{10} and x_{11}.

In the past few years, researchers have proposed many LCA-based semantics [3–6, 11], among which SLCA [8,11] is one of the most widely adopted semantics. Compared with LCA, SLCA [8,11] defines a subset of $LCA(Q)$, of which no LCA in the subset is an ancestor of any other LCA, which can be formally defined as $SLCA(Q) = \{v|v \in LCA(Q)$ and $\nexists v' \in LCA(Q)$, such that $v \prec_a v'\}$. In Fig. 1, although r, x_1 and x_8 are LCAs, r and x_1 are ancestors of x_4, and x_8 is an ancestor of x_9 and x_{10}, thus the set of SLCAs for $Q = \{a, b\}$ on D in Fig. 1 are x_4, x_5, x_9, x_{10} and x_{11}.

For SLCA computation, besides Stack [11], IL [11] and IMS [8] discussed in Section 1, the HashSearch (HS) [10] algorithm tries to skip as many useless nodes as possible by taking the shortest list L_1 as the working list. In each iteration, it sequentially picks a node v_1 from L_1, and computes a candidate SLCA node v based on v_1. To check whether v is an LCA node, for each keyword k_i, the HS algorithm uses a hash table H, which maintains, for each node v_i and each keyword k_i, the number of descendant nodes of v_i that directly contains k_i. Compared with the Stack, IL and IMS algorithm, even though HS looks better by comparing their time complexity, HS suffers from much more space to maintain the hash table. Essentially, HS sacrifices more space to achieve better efficiency. Moreover, for a given keyword query, although IL, IMS and HS can reduce the number of iterations by skipping many useless nodes, when the length of the shortest keyword inverted list is very long, their performance degenerates accordingly. Even if the length of the shortest list is not very long, when the number of results is much less than the length of the shortest list, all of them suffer from redundant computations, that is, the keyword distribution has great impact on their performances. As a comparison, our method takes the same set of keyword inverted lists as that of IL and IMS, which does not need additional space to maintain a big hash table as HS does, but achieves better performance gain even compared with HS. Another related work is [2], which proposed a new labeling scheme, i.e., JDewey labeling scheme, where each component of a JDewey label is a unique identifier among all nodes at the same tree depth. According to this property, the algorithm proposed in [2] computes all SLCA nodes based on set intersection operation on all lists of each tree depth from the leaf to the root.

2.3 Notations

Given a set of nodes S, the function $first(S)(last(S))$ returns the $smallest$ $(largest)$ node $v \in S$ such that $v \preceq_d v_i$ $(v_i \preceq_d v)$ for each $v_i \in S$. For example, consider

D in Fig. 1, let $L_a = \{a_1, a_2, ..., a_{401}\}$ be the inverted list of keyword a, we have $first(L_1) = a_1, last(L_1) = a_{401}$. In the following discussion, for a set of Dewey label lists $B = \{L_1, L_2, ..., L_m\}$, $L_i \in B$ is denoted as $B.L_i$.

3 List Partition

Definition 1. *A **block** is a set of Dewey label lists $B = \{L_1, L_2, ..., L_m\}$. A **partition** of B is a set of **blocks** $P_B = \{B_1, B_2, ..., B_n\}$, where each $B_j = \{L_1, L_2, ..., L_m\} \in P_B(j \in [1, n])$ satisfies the following conditions:*

1. *$\forall i \in [1, m]$, $B_j.L_i \subseteq B.L_i$,*
2. *$\exists i \in [1, m]$, such that $B_j.L_i \neq \emptyset$,*
3. *$\exists i \in [1, m]$, such that $B_j.L_i \subset B.L_i$,*
4. *(**Independence**) $\forall i \in [1, n], i \neq j, lca(B_i) \not\preceq_a lca(B_j)$ and $lca(B_j) \not\preceq_a lca(B_i)$,*
5. *(**Completeness**) $\forall i \in [1, m]$, $\forall v \in B.L_i$, if $lca(B) \prec_a v$, $v \in \bigcup_{j=1}^{n} B_j.L_i$.*

*If $\exists i{\in}[1, m]$, such that $B.L_i{=}\emptyset$, B is a **trivial block**; otherwise a **nontrivial block**.*

Intuitively, a block B is a set of Dewey label lists with at least one nonempty list, which corresponds to a subtree $T_{lca(B)}$ rooted $lca(B)$, while a partition $P_B = \{B_1, B_2, ..., B_n\}$ of B is a set of blocks corresponding to a set of subtrees $\{T_{B_1}, T_{B_2}, ..., T_{B_n}\}$ of $T_{lca(B)}$, where the root node $lca(B_i)$ of each subtree $T_{B_i}(i \in [1, n])$ is a descendant node of $lca(B)$, i.e., $lca(B) \prec_a lca(B_i)$. The *Independence* means that for any two blocks of a partition, the root nodes of their corresponding subtrees do not have AD relationship, thus all nodes of one block must strictly precede or succeed all nodes of the other block, that is, for a given partition $P_B = \{B_1, B_2, ..., B_n\}$, $\forall x, y {\in} [1, n], x \neq y$, $last(\bigcup_{i=1}^{m} B_x.L_i) \prec_d first(\bigcup_{i=1}^{m} B_y.L_i)$ or $last(\bigcup_{i=1}^{m} B_y.L_i) \prec_d first(\bigcup_{i=1}^{m} B_x.L_i)$. The *Completeness* means that for each $i \in [1, m]$, the union of the i^{th} list of all blocks in P_B covers all nodes in $B.L_i$ that are descendant of $lca(B)$. Since for a given partition P_B, each $B_i \in P_B(i \in [1, n])$ can be uniquely represented by $lca(B_i)$, henceforth, we use block $B_{lca(B_i)}$ to denote B_i if it belongs to a partition. According to Definition 1, all Dewey label lists of a given keyword query Q form a block B. B can have many possible partitions by recursively replacing some block $B_i \in P_B$ with P_{B_i}. Informally speaking, this means that these newly generated partitions are further fragmentations of P_B.

Example 3. For query $Q = \{a, b\}$ and the XML document D in Fig. 1, according to Definition 1, $B{=}\{\{a_1, ..., a_{401}\}, \{b_1, ..., b_{402}\}\}$ is a block, $P_B{=}\{B_{x_1}, B_{x_5}, B_{x_8}, B_{x_{11}}\}$ is a partition of B, where $B_{x_1} = \{\{a_1, ..., a_{200}\}, \{b_1, ..., b_{200}\}\}$, $B_{x_5}{=}\{\{a_{201}, ..., a_{301}\}, \{b_{201}, ..., b_{301}\}\}$, $B_{x_8} = \{\{a_{302}, ..., a_{400}\}, \{b_{302}, b_{303}\}\}$ and $B_{x_{11}}{=}\{\{a_{401}\}, \{b_{304}, ..., b_{402}\}\}$. Similarly, according to Definition 1, $P_{B_{x_1}} = \{B_{x_2}, B_{x_3}, B_{x_4}\}$, $P_{B_{x_5}} = \{B_{x_6}, B_{x_7}\}$, $P_{B_{x_8}}{=}\{B_{x_9}, B_{x_{10}}\}$, $P_{B_{x_{11}}}{=}\{B_{a_{401}}, B_{b_{304}}, ..., B_{b_{402}}\}$ are partitions of B_{x_1}, B_{x_5}, B_{x_8} and $B_{x_{11}}$, respectively. Obviously, B_{x_2}, B_{x_3}, B_{x_6}, B_{x_7}, $B_{a_{401}}, B_{b_{304}}, ..., B_{b_{402}}$ are trivial blocks, while $B_{x_1}, B_{x_4}, B_{x_5}, B_{x_8}, B_{x_9}, B_{x_{10}}$ and $B_{x_{11}}$ are nontrivial blocks.

Lemma 1. *Given a block B, if B is a trivial block, then SLCA computation on all nodes of B will not produce an SLCA node.*

Lemma 2. *Given a block B and one of its partitions $P_B = \{B_1, B_2, ..., B_n\}$, $\forall i \in [1, n], lca(B) \prec_a lca(B_i)$.*

Definition 2. *Given a block B, we say partition P_B is the maximum partition (**MPartition**) of B, denoted as P_B^M, if there does not exist another partition P'_B of B, such that some block $B_i \in P_B$ is a block of partition $P_{B'_j}$, where $B'_j \in P'_B$.*

Intuitively, the MPartition P_B^M of B is the one that consists of a set of blocks corresponding to the set of subtrees by removing the root node of $T_{lca(B)}$. Obviously, each block B has just one MPartition, and the rest partitions of B can be generated by recursively replacing some block of P_B^M with its MPartition.

Definition 3. *Given a nontrivial block B, we say B is a minimum nontrivial block (**MNBlock**) if no block of its MPartition is a nontrivial block.*

Lemma 3. *The LCA node of each MNBlock w.r.t Q is an SLCA node of Q.*

Definition 4. *Given a nontrivial block B and its MPartition $P_B^M = \{B_1, B_2, ..., B_n\}$, let $P_B^t \subseteq P_B$ be the set of trivial blocks of P_B. The equivalent block set **(EBSet)** $\mathcal{B}_B^E$ of B is recursively defined in formula 1.*

$$\mathcal{B}_B^E = \begin{cases} \{B\}, & P_B^t = P_B^M \\ P_B^M - P_B^t, & otherwise \end{cases} \tag{1}$$

Intuitively, for a given nontrivial block B, the *equivalence* lies in that B and its EBSet will produce the same set of SLCA nodes. If no block in B's MPartition is a nontrivial block, B is an MNBlock. According to Lemma 3, the only SLCA node of B w.r.t Q is $lca(B)$, thus its EBSet is itself; otherwise, its EBSet consists of all nontrivial blocks of its MPartition, since in such a case, $lca(B)$ must not be an SLCA node.

Lemma 4. *Given a nontrivial block B and one of its EBSet $\mathcal{B}_B^E$, we have:*

1. *if $\mathcal{B}_B^E = \{B\}$, $SLCA(B) = lca(B)$; otherwise,*
2. *$SLCA(B) = SLCA(\mathcal{B}_B^E) = \bigcup_{i=1}^{|\mathcal{B}_B^E|} SLCA(B_i)$, where $B_i \in \mathcal{B}_B^E$.*

Lemma 4 provides us a useful property to recursively prune useless blocks and get SLCA nodes as early as possible. That is, when processing a nontrivial block B_i, if B_i is an MNBlock, then $SLCA(B_i) = lca(B_i)$; otherwise, we just need to recursively process all blocks in its EBSet, i.e., all trivial blocks in its MPartition are discarded without being processed.

Theorem 1. *Given a keyword query Q, let B be a block consisting of the set of initial Dewey label lists, $\mathcal{B}_B^E = \{B_1, B_2, ..., B_n\}$ be the EBSet consisting of MNBlocks, which is generated by recursively replacing each non-MNBlock with the corresponding EBSet, then we have $SLCA(Q) = \{lca(B_i) | i \in [1, n]\}$.*

Example 4. According to Definition 4, we know $\mathcal{B}_B^E$=$\{B_{x_1}, B_{x_5}, B_{x_8}, B_{x_{11}}\}$ is an EBSet of $Q = \{a, b\}$ on D in Fig. 1. Since B_{x_1} and B_{x_8} are not MNBlocks, we recursively replace them by their EBSets until each block of $\mathcal{B}_B^E$ is an MNBlock. Specifically, we replace B_{x_1} by B_{x_4}, replace B_{x_8} by B_{x_9} and $B_{x_{10}}$, then we get an EBSet of B consisting of all MNBlocks, that is, $\mathcal{B}_B^E = \{B_{x_4}, B_{x_5}, B_{x_9}, B_{x_{10}}, B_{x_{11}}\}$. According to Theorem 1, we have $SLCA(Q) = \{lca(B_{x_4}), lca(B_{x_5}), lca(B_{x_9}), lca(B_{x_{10}}), lca(B_{x_{11}})\} = \{x_4, x_5, x_9, x_{10}, x_{11}\}$.

4 The Algorithm for SLCA Computation

From Theorem 1, we know that for a given query Q, the task of SLCA computation on the set of initial Dewey label lists can be equivalently transformed into recursively finding an EBSet that consists of MNBlocks. As shown by the name of Algorithm 1, "LP" means list partition. For a given query $Q = \{k_1, k_2, ..., k_m\}$, we firstly construct the initial block B (in line 1) that consists of the set of initial Dewey label lists of Q, then assign Dewey label "1" to $B.CP$ (in line 2), where $B.CP$ is a common prefix of all Dewey labels of B. In line 3, we recursively compute a partition of B by calling the findPartition procedure. We explain Algorithm 1 using the following example for limited space.

Algorithm 1: LPSLCA(Q) /*$Q = \{k_1, ..., k_m\}$*/

```
1  $B \leftarrow \{L_1, L_2, ..., L_m\}$ /*$L_i (i \in [1, m])$ is the Dewey label list of $k_i \in Q$*/
2  $B.CP \leftarrow 1$
3  findPartition(B)

4  Procedure findPartition($B$)  /*$B = \{L_1, L_2, ..., L_m\}$*/
5  $L_{min} \leftarrow$ minarg$_i\{|B.L_i|\}$; $n \leftarrow 0$
6  foreach (distinct component $x$ in the $(|B.CP| + 1)^{th}$ position of all Dewey labels of $L_{min}$) do
7      if ($\forall j \in [1, m], j \neq min, \exists l \in B.L_j$, such that $l[(|B.CP| + 1)] = x$) then    /*$l$ is a Dewey label*/
8          $n \leftarrow n + 1$
9          $B'.CP \leftarrow B.CP.x$ /*$B.CP$ is a Dewey label denoting a common prefix of all Dewey labels of $B$*/
10         foreach ($j \in [1, m]$) do
11             get the start (end) position of $B'.L_j$ according to $x(x + 1)$ by index lookup on $B.L_j$
12         endfor
13         findPartition($B'$)
14     endif
15 endfor
16 if ($n = 0$) then output $B.CP$ as an SLCA node        /*$n = 0$ means $\mathcal{B}_B^E = \{B\}$*/
```

Example 5. For query $Q = \{a, b\}$ and the XML document D in Fig. 1, according to algorithm 1, $B = B_r$ initially. The first block identified by findPartition is B_{x_1} by calling findPartition(B_r). Since B_{x_1} is not a trivial block, we recursively process B_{x_1} by calling findPartition(B_{x_1}) in line 13. When processing B_{x_1}, our method directly skips B_{x_2}, since it is a trivial block and does not satisfy the conditions of line 7. Although B_{x_3} is also a trivial block, our method will not even consider it since the blocks considered by Algorithm 1 are those generated by all distinct components of Dewey labels in the shortest list of each block. For B_{x_1}, we use components in the third level of Dewey labels in $B_{x_1}.L_a$ to generate blocks, thus as shown in Fig. 1, the next block generated after B_{x_2} is B_{x_4}, which is denoted as an enclosed number. And in this iteration, we need to recursively process B_{x_4} by calling findPartition$(\underline{B_{x_4}})$, during which we generate 100 trivial blocks, i.e., $B_{a_{101}}$, $B_{a_{102}}$, ..., $B_{a_{200}}$, according to all distinct components in the fourth level of Dewey labels in $B_{x_4}.L_a$. findPartition$(\underline{B_{x_4}})$ is stopped after outputting $B_{x_4}.CP$, i.e., $lca(B_{x_4}) = x_4$, as an SLCA node in line 16. After that, findPartition$(\underline{B_{x_1}})$ is also stopped. The second nontrivial block found in findPartition$(\underline{B_r})$ is B_{x_5}, then findPartition$(\underline{B_{x_5}})$ is invoked to find more blocks of its partition, during which block B_{x_6} will be skipped since it is a trivial block. Similar to B_{x_3}, our method will not generate B_{x_7} since blocks in B_{x_5}'s partition are generated by all distinct components in the third level of Dewey labels in $B_{x_5}.L_a$. In line 16, findPartition$(\underline{B_{x_5}})$ outputs $B_{x_5}.CP$, i.e., $lca(B_{x_5}) = x_5$, as an SLCA and then stops. The following processing is similar, we find B_{x_8}, based on which we further find two MNBlocks, i.e., B_{x_9} and $B_{x_{10}}$, and output their LCA nodes, i.e., x_9 and x_{10}, as qualified results. In findPartition$(\underline{B_{x_9}})$, our method just generate one trivial block, i.e., $B_{b_{302}}$, because in this case, blocks are generated by components in the forth level of Dewey labels in $B_{x_9}.L_b$. Similarly, findPartition$(\underline{B_{x_{10}}})$ will also produce just one trivial block, i.e., $B_{a_{400}}$, which is generated by components in the forth level of Dewey labels in $B_{x_{10}}.L_a$. The last block found in findPartition$(\underline{B_r})$ is $B_{x_{11}}$, which is an MNBlock and thus we get the last SLCA node, i.e., $lca(B_{x_{11}}) = x_{11}$, in line 16 of findPartition$(\underline{B_{x_{11}}})$. In findPartition$(\underline{B_{x_{11}}})$, our method generates just one block, i.e., $B_{a_{401}}$, because blocks are generated by components in the third level of Dewey labels of $B_{x_{11}}.L_a$. Therefore the final SLCA nodes outputted by Algorithm 1 are x_4, x_5, x_9, x_{10} and x_{11}. In Fig. 1, all enclosed numbers denote the detailed processing steps.

Note that in Algorithm 1, we assume that for a given query $Q = \{k_1, k_2, ..., k_m\}$, the set of Dewey label lists satisfy $0 < |L_1| \leq |L_2| \leq ... \leq |L_m|$. The case where at least one Dewey label list is empty can be easily processed before line 1. According to Algorithm 1, our method *does not need to pre-load* all Dewey label lists into memory, since it *recursively* computes all MNBlocks in document order. Moreover, the operation of computing the LCA node for two nodes that is frequently used by existing methods can be avoided, since we have already got it when finding a nontrivial block in line 9 of Algorithm 1. Finally, unlike existing algorithms [8, 10, 11], the distribution of the underlying data imposes *little influence* on our method.

Now we analyze the complexity of LPSLCA. We begin by establishing an upper bound on the number of blocks generated by our LPSLCA algorithm.

Lemma 5. *For a given keyword query $Q = \{k_1, k_2, ..., k_m\}$, Algorithm 1 produces at most $d|L_1|$ blocks, where d is the depth of the given XML tree, L_1 is the shortest Dewey label list of Q.*

Proof. Assume that all blocks are computed by each level in a top-down way, as shown in line 5 to line 12 of Algorithm 1, each block is computed based on a distinct component x which comes from the shortest list of a block, and the sum of lengths of all shortest lists of blocks in the same level is not greater than the length of L_1. Thus in each level, the number of blocks computed by Algorithm 1 is bounded by $|L_1|$. Therefore, there are at most $d|L_1|$ blocks need to be computed, no matter whether they are trivial or nontrivial ones, where d is the depth of the given XML tree. ■

Consider the worst case where (1) there are $|L_1|$ qualified SLCA nodes, (2) all of them are in the d^{th} level of the given XML tree, and (3) Algorithm 1 produce $|L_1|$ nontrivial blocks in the second to the d^{th} level. In this case, there are no trivial block can be skipped by Algorithm 1. To identify the range of each list of a nontrivial block, we need to call the index lookup operation two times, which can be implemented using either binary or galloping search. Note that the comparison operation in binary or galloping search is not based on Dewey labels, but integers. Therefore, in the second level, the cost of computing a block is $O(m \log |L_m|)$. For other levels, the average length of the longest list of each block is $|L_m|/|L_1|$. Thus the overall time complexity is $O(m|L_1| \log |L_m| + dm|L_1| \log \frac{|L_m|}{|L_1|})$.

5 Experimental Evaluation

5.1 Experimental Setup

Our experiments were implemented on a PC with Intel(R) Core(TM) i5 M460 2.53 GHz CPU, 2 GB memory, and Windows XP professional as the operating system.

The algorithms used for comparison include the Stack [11], IL [11], IMS [8], JDewey [2] and HS [10] algorithms. All algorithms were implemented using Microsoft VC++, all results are the average time by executing each algorithm 100 times on hot cache.

Table 1. Statistics of keywords used in our experiment

Keyword	tissue	baboon	necklace	arizona	cabbage	hooks	shocks	patients	cognition	villages
$\lvert L_{Dewey}\rvert$	384	725	200	451	366	461	596	382	495	829
Keyword	male	takano	order	school	check	education	female	province	privacy	gender
$\lvert L_{Dewey}\rvert$	18441	17129	16797	23561	36304	35257	19902	33520	31232	34065
Keyword	bidder	listitem	keyword	bold	text	time	date	emph	incategory	increase
$\lvert L_{Dewey}\rvert$	299018	304969	352121	368544	535268	313398	457232	350560	411575	304752

We used XMark (582MB) and DBLP (876MB) datasets for our experiment. We have selected from XMark dataset 30 keywords classified into three categories according to their occurrence frequencies (i.e. $|L_{Dewey}|$ line in Table 1): (1) low frequency (100-1000), (2) median frequency (10000-40000), (3) high frequency (300000-600000). Based on these keywords, we generated 18 queries as shown in Table 2. We only present experimental results on XMark dataset for limited space.

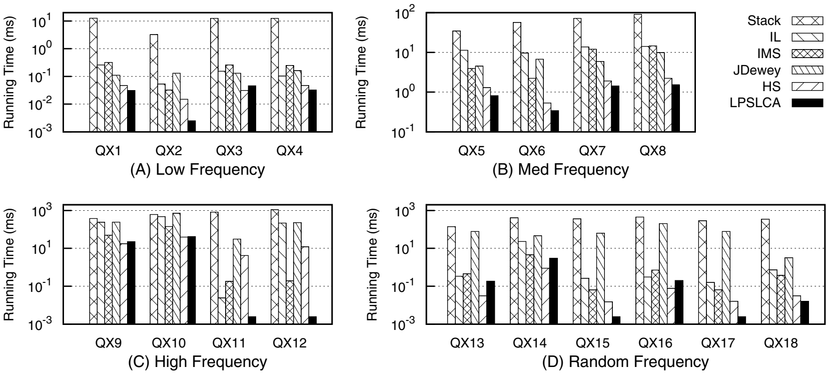
5.2 Performance Comparison and Analysis

For a given query, we define the *result selectivity* as the size of the results over the size of the shortest inverted list. Fig. 2 shows the running time of different algorithms for query QX1 to QX18, from which we know that our method outperforms existing methods for most of these queries, especially when result selectivity is low (see Table 2), such as QX2, QX6, QX11, QX12, QX15, QX17 and QX18. The reasons lie in: (1) the set of Dewey label lists may possess different keyword distributions, and our

Table 2. Queries on 582MB XMark dataset, $|L_{min}|$ denotes the length of the shortest Dewey label list for a query, N_S is the number of qualified SLCA results, $R_S = N_S/|L_{min}|$ denotes the result selectivity

ID	Keywords	$\|L_{min}\|$	N_S	$R_S(\%)$	Freq.
QX1	villages,hooks	461	9	1.95	Low
QX2	baboon,patients,arizona	382	1	0.26	
QX3	cabbage,tissue,shocks,baboon	366	9	2.46	
QX4	shocks,necklace,cognition,cabbage,tissue	200	9	4.5	
QX5	female,order	16700	570	3.41	Med
QX6	privacy,check,male	18428	29	0.16	
QX7	takano,province,school,gender	17129	107	0.62	
QX8	school,gender,education,takano,province	17129	107	0.62	
QX9	bold,increase	304706	34136	11.2	High
QX10	date,listitem,emph	304969	43777	14.35	
QX11	incategory,text,bidder,date	299018	1	0.0003	
QX12	bidder,date,keyword,incategory,text	299018	1	0.0003	
QX13	incategory,cabbage	366	224	61.2	Random
QX14	province,bold,increase	33520	427	1.27	
QX15	listitem,emph,arizona	451	1	0.22	
QX16	bold,increase,hooks,takano	461	6	1.3	
QX17	emph,arizona,villages,education	451	1	0.22	
QX18	check,bidder,date,baboon	742	1	0.13	

method can wisely prune useless keyword nodes according to their distribution, that is, computing sub-blocks according to nodes of the shortest list of *each* block; (2) the LCA operation frequently used in existing methods are avoided in our method.



(A) Low Frequency

(B) Med Frequency

(C) High Frequency

(D) Random Frequency

Fig. 2. Comparison of running time for SLCA computation on XMark dataset (log-scaled)

Besides, we have the following observations for existing methods: (1) IL and IMS are usually much better than Stack, because Stack is not able to skip useless nodes; (2) IL could perform better than IMS in some cases, such as QX1, QX3, QX4, QX11, QX13 and QX16, but the performance gain is usually much less than that of IMS on IL, such as QX5, QX6, QX14 and especially QX12, because for these queries, the set of Dewey label lists for each one has different keyword distributions, and IL is not as flexible as IMS on utilizing various keyword distributions to accelerate the computation; (3) JDewey could perform better than Stack in many cases, but is usually beaten by IL and IMS, such as QX13 to QX18, because JDewey needs to process all lists of each level from the leaf to the root; and for all lists of each level, after finding the set of common nodes, it needs to recursively delete all ancestor nodes in all lists of higher levels, which is very expensive in practice; (4) by pre-recording the containment relationship between each node and each keyword in a hash table, HS is very efficient for most queries as compared with other existing methods, but it is not as efficient as our method when the result selectivity is low, especially for QX11 and QX12. When the result selectivity becomes high, HS can be better than our method, such as QX3, QX9, QX10, QX13, QX14 and QX16.

6 Conclusions

Considering the *variety* of keyword distributions and the *inflexibility* of existing methods in choosing an appropriate processing strategy for SLCA computation, we propose an efficient algorithm, namely LPSLCA, which processes all nodes in the set of inverted Dewey label lists in a *top-down* way based on list partition to accelerate the SLCA computation. Our method *recursively* computes the set of *minimum nontrivial block* and outputs their LCA nodes as qualified results, during which it can quickly prune useless nodes based on the distribution of all nodes in each given block so as to realize *fine-grained optimization*. Experimental results verify the performance advantages of our method according to various evaluation metrics.

Acknowledgment. This research was partially supported by grants from the Natural Science Foundation of China (No. 61073060, 61040023, 61103139) and the Fundamental Research Funds of Hebei Province (No. 10963527D). Zhifeng Bao was partially supported by the Singapore Ministry of Education Grant No. R252-000-394-112 under the project name of UTab.

References

1. Bao, Z., Ling, T.W., Chen, B., Lu, J.: Effective xml keyword search with relevance oriented ranking. In: ICDE, pp. 517–528 (2009)
2. Chen, L.J., Papakonstantinou, Y.: Supporting top-k keyword search in xml databases. In: ICDE, pp. 689–700 (2010)
3. Cohen, S., Mamou, J., Kanza, Y., Sagiv, Y.: Xsearch: A semantic search engine for xml. In: VLDB, pp. 45–56 (2003)
4. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: Xrank: Ranked keyword search over xml documents. In: SIGMOD Conference, pp. 16–27 (2003)
5. Li, G., Feng, J., Wang, J., Zhou, L.: Effective keyword search for valuable lcas over xml documents. In: CIKM, pp. 31–40 (2007)
6. Li, Y., Yu, C., Jagadish, H.V.: Schema-free xquery. In: VLDB, pp. 72–83 (2004)
7. Liu, Z., Chen, Y.: Identifying meaningful return information for xml keyword search. In: SIGMOD Conference, pp. 329–340 (2007)
8. Sun, C., Chan, C.Y., Goenka, A.K.: Multiway slca-based keyword search in xml data. In: WWW, pp. 1043–1052 (2007)

9. Tatarinov, I., Viglas, S., Beyer, K.S., Shanmugasundaram, J., Shekita, E.J., Zhang, C.: Storing and querying ordered xml using a relational database system. In: SIGMOD Conference, pp. 204–215 (2002)
10. Wang, W., Wang, X., Zhou, A.: Hash-Search: An Efficient SLCA-Based Keyword Search Algorithm on XML Documents. In: Zhou, X., Yokota, H., Deng, K., Liu, Q. (eds.) DASFAA 2009. LNCS, vol. 5463, pp. 496–510. Springer, Heidelberg (2009)
11. Xu, Y., Papakonstantinou, Y.: Efficient keyword search for smallest lcas in xml databases. In: SIGMOD Conference, pp. 537–538 (2005)
12. Zhou, R., Liu, C., Li, J.: Fast elca computation for keyword queries on xml data. In: EDBT, pp. 549–560 (2010)

Efficiently Identifying Contributors for XML Keyword Search*

Ya-Hui Chang, Po-Hsien Chien, and Yu-Kai Chang

Department of Computer Science and Engineering,
National Taiwan Ocean University
yahui@ntou.edu.tw

Abstract. Keyword search is a friendly mechanism for users to obtain desired information in XML databases. Among all the research efforts on automatically reasoning the meaningful answers for users, the MaxMatch System [6] proposed the concept of *contributor* and has attracted a lot of attention. In this paper, we propose the TDPrune system to improve the efficiency of identifying contributors. It is mainly achieved by avoiding processing unnecessary nodes, and also by visiting nodes only once instead of twice. According to the experimental results, our TDPrune system usually outperforms the MaxMatch system, and the difference of execution time is sometimes by an order of magnitude.

1 Introduction

Keyword search provides a convenient interface for users to obtain desired information from XML documents, but irrelevant data may be returned due to lacking exact query semantics. Therefore, there are a lot of researches on automatically reasoning meaningful answers for users.

In general, an XML document could be viewed as a rooted tree, where each node represents an element or contents. The LCA-based approaches will identify *the LCA node* first, which contains every keyword under its subtree at least once [1,2,3,4,5,6,7,8,9]. Since the LCA nodes sometimes are not very *specific* to users' query, the researchers in [8] proposed the concept of *SLCA (smallest lowest common ancestor)*, where a node is said to be an SLCA if (i) it is an LCA, and (ii) it has no descendant nodes that also contain all query keywords. For example, consider the XML tree in Figure 1, where each node is associated with a unique Dewey number. For the query $Q_1 = ($*Jim, POSITION, TEAM_NAME*$)$, the LCA list is [1, 1.2, 1.3]. Since node 1 has descendant nodes 1.2 and 1.3 that are also LCAs, only nodes 1.2 and 1.3 are SLCAs. We could see that the two corresponding *TEAM* elements contain more specific information than the element *LEAGUE*, which is node 1.

The SLCA approach achieves *specificity* based on the ancestor/decendant relationship, but they do not distinguish the importance of sibling nodes. Therefore,

* This work was partially supported by the National Science Council under Contract No. NSC 100-2321-B-019 -004 -.

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 185–196, 2012.
© Springer-Verlag Berlin Heidelberg 2012

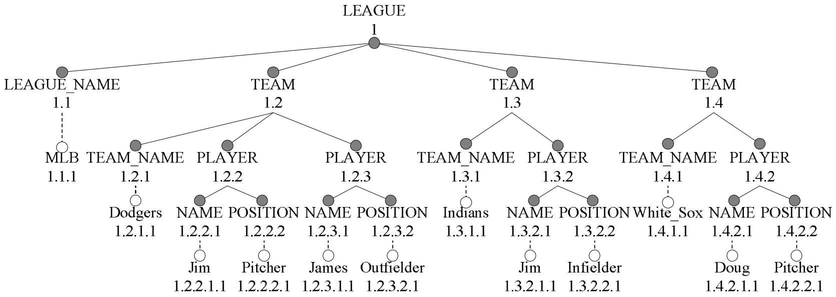


Fig. 1. A sample XML tree

the researchers in [6] further proposed the concept of *contributors*, where a node is a contributor if it represents more (or equal to) keywords compared with its sibling nodes, and only contributors will be returned. For example, consider Q_1 and the subtree rooted at node 1.2 in Figure 1. Since the subtree rooted at node 1.2.2 contains keywords *Jim* and *POSITION*, and the subtree rooted at node 1.2.3 only contains the keyword *POSITION*, the latter subtree is *pruned* by the former subtree.

The authors in [6] gave an efficient system *MaxMatch* to produce the desired output, which consists of a core module called *PruneMatch* to prune less important nodes. The main idea is first to collect the keywords each subtree represents in a postorder traversal, and then use this information to determine which nodes to be pruned in a preorder traversal. Although MaxMatch has been shown quite efficient, its performance can be further improved. In this paper, we propose a top-down approach, and the corresponding system is called *TDprune*. The main difference with MaxMatch is that our approach visits nodes only once instead of twice. Further, during the tree traversal, we manage to examine only required nodes, and hence process less nodes than MaxMatch in most cases. We have designed a series of experiments to compare the performance of two systems. The results show that our approach usually outperforms the MaxMatch system, and the execution time sometimes differs by an order of magnitude.

The rest of this paper is organized as follows. In Section 2, we briefly introduce the MaxMatch system. We then present the main data structures used in our system in Section 3. The core algorithms are discussed in Section 4. We further discuss the experimental studies in Section 5. Finally, Section 6 concludes this paper.

2 Preliminaries

In the following, we formally deliver the definitions and notation given in MaxMatch [6], and then introduce its component algorithms. The sample XML tree given in Figure 1 will be used in the running examples throughout this paper.

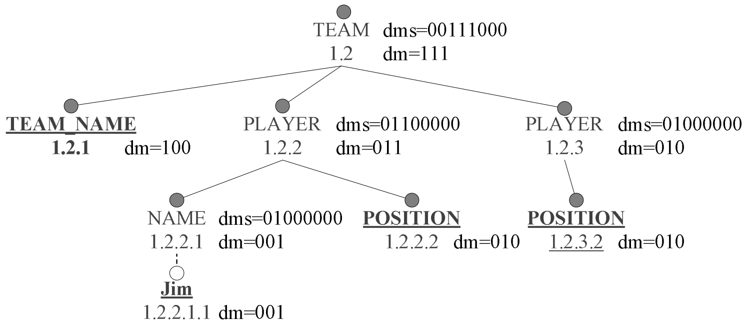


Fig. 2. Sample dMatch and dMatchSet arrays

2.1 Definitions

Definition 1 [Match]: A node is a *match* if its tag name or the content corresponds to a given query keyword. □

Definition 2 [Descendant Match]: The *descendant matches* of a node n, denoted as $dMatch(n)$ or $n.dMatch$, are a set of query keywords, each of which has at least one match in the subtree rooted at n. $dMatch(n)$ could also be seen as a bit array of size w or a decimal value for simplicity, where w is the number of keywords. Each keyword corresponds to its own bit. □

Definition 3 [Contributor]: A node n is a *contributor* if (i) n is the descendant of a given SLCA or n itself is one of the SLCAs, and (ii) n does not have a sibling n_2 such that $dMatch(n_2) \supset dMatch(n)$. □

Definition 4 [dMatchSet]: The *dMatchSet* value of a node is a bit array of size 2^w to record the *dMatch* values of its children. All the bits are initialized as $false$ at the beginning. The j^{th} bit, $j \leq 2^w - 1$, is set to *true* if it has at least one child n_c such that $dMatch(n_c) = j$. □

Definition 5 [Relevant Match]: A match node n is considered *relevant* if (i) n has an ancestor-or-self t such that t is one of the SLCAs, and (ii) each node on the path from n to t is a contributor. □

Definition 6 [Query Result]: Given an XML tree and a set of the query keywords, the *query result* is defined as all of the relevant matches (including value children, if any) contained in the subtrees rooted at the SLCAs. The paths from the SLCA to each relevant match will be also output. □

Example 1. Recall that Q_1 = (*Jim, POSITION, TEAM_NAME*). We show its corresponding dMatch arrays (noted as *dm*) and dMatchSet arrays (noted as *dms*) for part of the sample XML tree in Figure 2. Here the keywords (from left to right) correspond to the first (right-most), the second, and the third bits of

```
Algorithm IsContributor
Input: node $n$
Output: boolean
for $n_p \leftarrow n.parent$
$i = |n.dMatch|$
for $j \leftarrow i + 1$ to $2^w - 1$ do
  if $n_p.dMatchSet[j] = true$ and $AND(i, j) = i$ then
    return $false$
  end if
end for
return $true$
```

Fig. 3. Algorithm *IsContributor*

$dMatch$. Observe that $dMatch(1.2.2) = 011_{bin}$ and $dMatch(1.2.3) = 010_{bin}$. By Definition 3, nodes 1.2.3 is not a contributor since $dMatch(1.2.3)$ is a proper subset of $dMatch(1.2.2)$. The functionality of dMatchSet will be explained later when we show the corresponding algorithm. □

2.2 MaxMatch

The MaxMatch system [6] could be decomposed into four modules as follows:

1. Retrieve the matches of each query keyword.
2. Compute the SLCAs by the algorithm given in [8].
3. Group all matches according to their SLCA ancestors.
4. For each SLCA, construct the correct value of $dMatch$ and $dMatchSet$ for each node from the matches to the SLCA. Determine the contributors in the preorder traversal sequence by procedure $IsContributor$ (Figure 3).

The functionality of the first three modules are pretty clear. We now explain the fourth module, called *PruneMatch* in more detail. Assume that the third module outputs a set of groups. Each group $\{t, M\}$ consists of an SLCA node t and the sorted matches M under the subtree rooted at t. The fourth module performs the following two functions: (i) constructing the tree for each group based on the Dewey encodings, and (ii) pruning the non-contributors. First, the construction proceeds in a manner of postorder tree traversal. Each constructed tree is composed of the nodes on the paths from every match m up to the SLCA node t. For each node n from m (exclusively) up to t, the j^{th} bit of $n.dMatch$ is set to 1 if n matches the j^{th} keyword. Besides, $n.dMatch$ is further updated according to the bitwise OR operator with $n_c.dMatch$, and $n_p.dMatchSet[n.dMatch]$ is set to $true$, *i.e.*, the value 1, where n_p and n_c are the parent and the child of n, respectively. As shown in Figure 2, dMatchSet(1.2)=00111000_{bin} since the dMatch values of its child nodes 1.2.3, 1.2.2, 1.2.1 are 2, 3, 4, respectively. That is, the dMatchSet array of a node summarizes the dMatch values of all its child nodes.

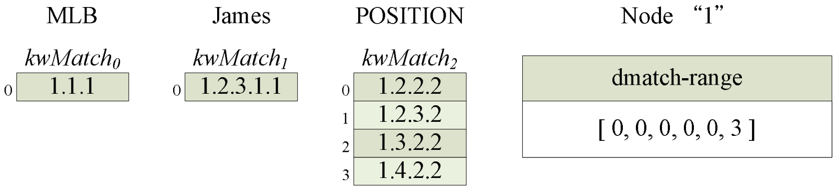


Fig. 4. Sample kwMatch and dmatch-range structures

After constructing the tree, MaxMatch performs a preorder tree traversal to identify the contributors by procedure *IsContributor*, as listed in Figure 3. It takes a node n as the input, and determines whether it is a contributor or not. The **for** loop starts from $i + 1$ instead of 1 because only those siblings whose $dMatch$ values are larger than i are possible to prune the node n. Besides, $n_p.dMatchSet[j] = true$ indicates that n has a sibling whose $dMatch$ value is j. If bitwise $AND(i, j) = i$, it is obvious that i is the proper subset of j. Take query Q_1 as an example. Recall that the values of $dMatch(1.2.2)$ and $dMatch(1.2.3)$ are 011_{bin} and 010_{bin}, respectively. Since $dMatch(1.2.3) < dMatch(1.2.2)$ and $AND(dMatch(1.2.3), dMatch(1.2.2)) = dMatch(1.2.3)$, node 1.2.3 is eventually pruned by node 1.2.2.

The *IsContributor* procedure can efficiently determine which child nodes to be pruned for a given node. However, the PruneMatch algorithm needs to totally traverse the tree twice. In the following, we will discuss how we combine the two steps into one step to expedite the processing.

3 Data Structure

In this section, we explain the main data structures used in our TDprune system. We will use the following query as the running example hereafter: Q_2 = *(MLB, James, POSITION)*. Note that this query only corresponds to one SLCA node, which is the root node 1.

3.1 kwMatch

As the first module of MaxMatch, TDprune needs to retrieve the matches of each query keyword in the first beginning. In our system, the matches are originally represented in the database. By using a B-tree index with each keyword as the key, the matches corresponding to the i_{th} query keyword will be retrieved and represented in kwMatch$_i$. Each kwMatch$_i$ is an integer array with the dimensions r_i and c_i, where r_i is the number of corresponding matches, and c_i is the largest number of components for all Dewey encodings. For the m_{th} match corresponding to the i_{th} keyword, its n_{th} component value of the Dewey encoding will be represented in kwMatch$_i$[m, n]. Note that the matches are represented in kwMatch in sorted order. The three kwMatch structures corresponding to Q_2 and the sample XML tree are shown on the left part of Figure 4.

3.2 Dmatch-Range

Recall that the second and third modules of MaxMatch identify all the SLCA nodes and its descendant matches. In MaxMatch, it uses a linked list of nodes to represent each SLCA group. In contrast, we associate each node a data structure called *dmatch-range*, or *DR* in short, to achieve the same functionality. Suppose the input query consists of w keywords. Each *dmatch-range* is an integer array with dimension $2w$. Consider node n and the i_{th} keyword, $n.DR[2i]$ and $n.DR[2i+1]$ will represent the first position and the last position of n's descendant nodes in kwMatch$_i$, respectively.[1] The correctness of such representation is justified by the following lemma:

Lemma 1. *Consider the array A which consists of a list of sorted Dewey encodings. For a node v, the descendant nodes of v in A, if any, will be represented in continuous positions of A.*

Proof: The sorted order of Dewey encodings correspond to the preorder sequence of XML trees. In preorder traversal, all descendant nodes of a particular node will be continuously visited. □

Note that we can efficiently obtain the values of dmatch-range, since it is pretty simple to determine the ancestor/descendant relationship based on Dewey encoding. As shown on the right of Figure 4, the DR array of node 1 covers all positions of the three kwMatch structures, since all nodes are descendants of node 1. As another example, suppose node 1.2 is an SLCA and consider the $2nd$ keyword "POSITION". Since kwMatch$_2$[0] = 1.2.2.2 and 1.2 is the prefix of 1.2.2.2, node 1.2.2.2 represents a descendant of node 1.2. Similarly, kwMatch$_2$[1] = 1.2.3.2 is a descendant node, but kwMatch$_2$[2] = 1.3.2.2 is not a descendant node. By Lemma 1, we can determine that the value of DR[2, 3] for node 1.2 is $[0, 1]$.

By utilizing the information in kwMatch, we can easily show each SLCA group by properly assigning the dmatch-range array for an SLCA node. Moreover, the dmatch-range array for a non-SLCA node can be used to infer its dMatch value. We will explain this point further in the next section.

4 Algorithms

In this section, we will explain how our TDPrune system works and discuss the core algorithms. The main idea of our approach is to traverse the tree from top to bottom, and output a node as soon as it is identified as a contributor. In the following subsections, we first present a supporting algorithm called FindNextChild, and then discuss how our PruneMatch module works.

[1] Note that the keyword sequence and the array index both start from 0.

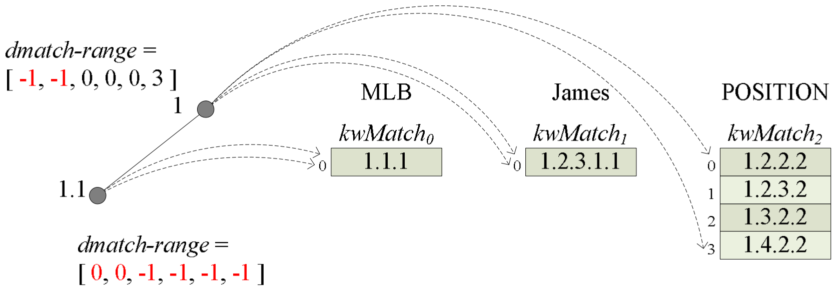


Fig. 5. The first invocation of Algorithm FindNextChild for Node 1

4.1 Algorithm FindNextChild

Recall that a contributor is determined based on comparing the dMatch values among all sibling nodes. For a parent node n, Algorithm FindNextChild is designed to create the child node n_c along with its dMatch value, where n_c has the smallest Dewey encoding among all unproduced child nodes. That is, by repeating invoking Algorithm FindNextChild, n's child nodes will be produced in preorder sequence. Note that we use the dmatch-range structure to infer the dMatch value of a node. Consider a node n. Initially, we assign all component values of the associated dmatch-range array as "-1". For the i_{th} keyword, we then access kwMatch$_i$ and let $n.DR[2i]$ and $n.DR[2i+1]$ represent the first position and the last position of its descendant nodes in kwMatch$_i$. If $n.DR[2i]$ is not negative after the process, node n must have at least one descendant node which matches the i_{th} keyword. We can therefore use this information to obtain the full dMatch array of node n.

Another point to note is that Algorithm FindNextChild can efficiently identify the child node with the smallest encoding based on the structures kwMatch and dmatch-range. Suppose that the encoding of the parent node n consists of l components. Then, all child nodes of n will have $l + 1$ components, and the last components can be easily obtained from the kwMatch arrays. Also, since $n.DR[2i]$ points to the first position of n's descendant nodes in kwMatch$_i$, and kwMatch presents encodings in increasing order, we can infer that $n.DR[2i]$ will represent the smallest encoding among all its descendant nodes in kwMatch$_i$. Therefore, as the same idea of the Merge-Sort algorithm, to find the child with the smallest encoding, we only need to compare all the encodings pointed by $n.DR[2i]$, $0 \leq i < w$.

Due to space limitation, we do not list the complete algorithm. Instead we use an example to illustrate how it works. Consider query Q2, whose corresponding kwMatch structures and the dmatch-range of node 1 are shown in Figure 4. From the values of $DR[0]$, $DR[2]$, and $DR[4]$, we identify that the smallest encoding among the child nodes of node 1 is 1.1, so we output node 1.1 along with its dmatch-range and update the dmatch-range of node 1, as shown in Figure 5. In that figure, we also use the dotted arrow to show the range in kwMatch covered

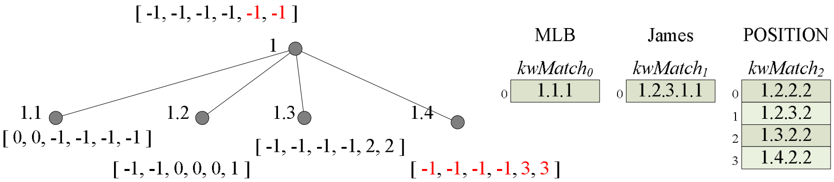
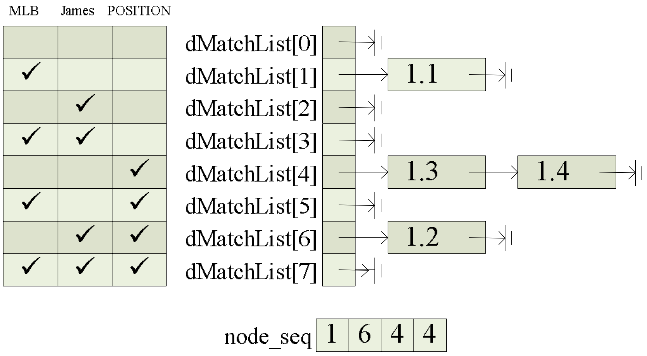


Fig. 6. The fourth invocation of Algorithm FindNextChild for Node 1



MLB	James	POSITION
✓		
	✓	
✓	✓	
		✓
✓		✓
	✓	✓
✓	✓	✓

Fig. 7. A sample dMatchList

by the dmatch-range structure. We can clearly see that node 1.1 has descendant nodes only in kwMatch$_1$, while node 1 still has unprocessed descendant nodes in kwMatch$_2$ and kwMatch$_3$. After three more iterations, we produce all child nodes of node 1 along with their dmatch-range arrays, as shown in Figure 6. Observe that now the dmatch-range array of node 1 has all the component values as -1.

4.2 PruneMatch

In this subsection, we discuss the PruneMatch module in our system. As discussed in Section 2, the MaxMatch system first builds the complete tree structure, and uses the structure dMatchSet to determine which child node is a contributor. In our system, we do not explicitly construct the tree structure. Instead, for each parent node n, we construct a *dMatchList* structure, which is an array of node lists, to represent its child nodes based on the state of the corresponding dMatch array. Specifically, suppose there are w query keywords. The dimension of the dMatchList array is 2^w, corresponding to all possible combinations of query keywords. Each dMatchList component represents a linked list of child nodes of n which have the same dMatch value. The dMatchList for Q_2 and the sample XML tree is shown in Figure 7, where the corresponding dMatch value of each dMatchList component is shown on the left. Note that there is one more structure shown on the bottom of Figure 7, which is *node_seq*. It is an integer array representing which dMatchList

Algorithm PruneMatch
Input: node n
Output: the part of the answer tree rooted at n

```
 1: output $n$; $nc\_cnt \leftarrow$ 0
 2: for each $n_c$ produced by FindNextChild($n$) do
 3:   for each $i_{th}$ query keyword do
 4:     if $n_c.DR[2*i]$ != -1 then
 5:       set the $i_{th}$ bit of $dMatch$ to 1
 6:     end if
 7:   end for
 8:   append $n_c$ into $n.dMatchList[num(n_c.dMatch)]$; $node\_seq[nc\_cnt] \leftarrow$
      $num(n_c.dMatch)$; $nc\_cnt$++;
 9: end for
10: if $nc\_cnt$ = 0 then
11:   do nothing
12: else if $nc\_cnt$ = 1 then
13:   $v \leftarrow$ get the only node from $n.dMatchList[n_c.dMatch]$
14:   PruneMatch($v$)
15: else
16:   for each int $i$ in $node\_seq$ do
17:     $is\_contributor \leftarrow true$
18:     for int $j \leftarrow i$ + 1; $j < 2^w$; $j$++ do
19:       if $n.dMatchList[j].size >$ 0 and $AND(i, j)$ = $i$ then
20:         $is\_contributor \leftarrow false$; break;
21:       end if
22:     end for
23:     if $is\_contributor$ = $true$ then
24:       $v \leftarrow$ get the first node from $n.dMatchList[i]$; PruneMatch($v$);
25:     end if
26:   end for
27: end if
```

Fig. 8. Algorithm *PruneMatch*

component a node is appended into. Particularly, if node_seq$[i] = j$, it means that the i_{th} node is represented in dMatchList$[j]$. This structure is used to assist us in outputting nodes in preorder sequence, as will be shown later.

Algorithm PruneMatch is listed in Figure 8. The variable *nc_cnt* is used to represent the sequence number of each produced child node, and the final value represents how many child nodes the node n has. Besides, *dMatch* is a binary number, and the function *num* converting a binary number to a decimal number. This algorithm first outputs the input node n. It then uses the **for** loop in L2-L9 to produce a child node n_c of n, obtain its dMatch value, and put n_c in the correct component of dMatchList. The following actions are based on how many child nodes are produced. If there is only one, which is definitely a contributor, we invoke Algorithm PruneMatch recursively on this node (L13-L14). Otherwise, we need to determine if a child node is a contributor. The processing sequence

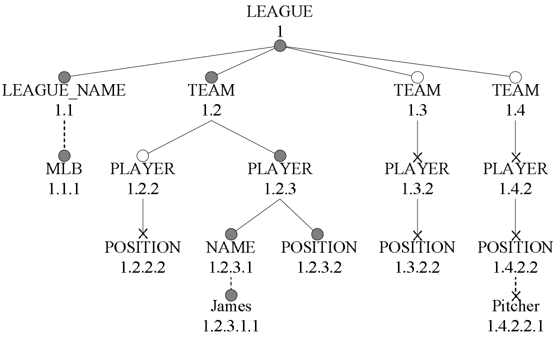


Fig. 9. Classification of nodes

is based on node_seq (L16), and the **for** loop in L18-L22 performs the same test as in MaxMatch (see Figure 3). If a child node is identified as a contributor, we will invoke Algorithm PruneMatch recursively on it to process its descendant nodes (L23-L25).

4.3 Analysis

We conclude this section by discussing the nodes processed by TDPrune and MaxMatch, respectively. Denote V as the set of all match nodes and their ancestors. Nodes in V can be classified into the following three groups:

- v_c: node itself and its parent node are both contributors
- v_{pc}: node itself is not a contributor, but its parent node is a contributor
- v_{pnc}: the node's parent node is not a contributor

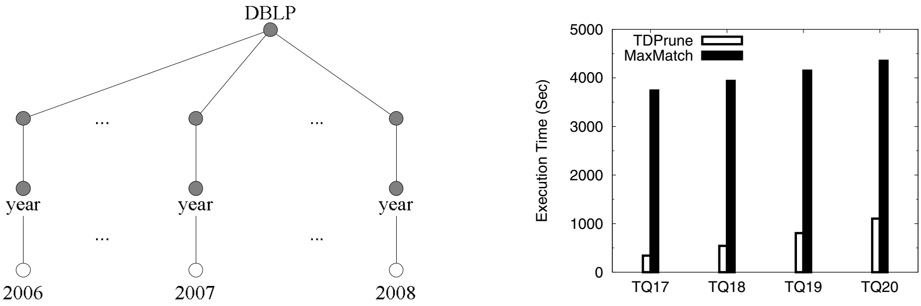
Consider Query Q2. Its corresponding classification of nodes is shown in Figure 9, where nodes v_c, v_{pc} and v_{pnc} are represented by solid circles, hollow circles, and crosses, respectively. For example, node 1.2 is a v_c, since it and its parent node 1 are both contributors. Node 1.3 is a v_{pc} since itself is not a contributor. This also makes node 1.3.2 a v_{pnc}.

Recall that the PruneMatch module of MaxMatch consists of two steps. First, it visits all nodes in V to set the value of dMatch and dMatchSet. Second, it examines nodes v_c and v_{pc} to determine contributors. In contrast, the PruneMatch module of our TDPrune system directly produces nodes v_c and v_{pc} and determine which are contributors. We can clearly see that MaxMatch additionally creates nodes v_{pnc} compared with our system, and the ratio of processed nodes can be shown by the following equation:

$$contribution\ rate = \frac{|v_c| + |v_{pc}|}{|V|} \tag{1}$$

Table 1. Test queries

No	Dataset	Keywords	Contribution Rate
TQ17	DBLP	dblp, year, 1995	1.9%
TQ18	DBLP	dblp, year, 2009	9.9%
TQ19	DBLP	dblp, year, 2009, 2010	18.1%
TQ20	DBLP	dblp,year, 2006, 2007, 2008	25.8%



(a) Output

(b) Execution time

Fig. 10. Effects of contribution rates

We will use the contribution rate later in Section 5 to examine the performance of two systems.

5 Experiments

In this section, we compare the performance of the TDPrune system with the MaxMatch system. The experiments are performed on a personal computer with Intel Quad-Core 3.4 GHz CPU and 16GB memory, with the Microsoft Windows 7 operating system.

We first apply the Baseball dataset[2] and the Mondial dataset[3] to perform the experiments. Sixteen test queries are adopted from the MaxMatch paper [6], which were designed to cover a variety of cases. The experimental results, which are omitted due to space limitation, show that TDPrune is more efficient than MaxMatch for all of those queries. In some cases, the difference is even by an order of magnitude when the contribution rates are very low.

We then apply the DBLP dataset[4] to examine the effect of the contribution rate further. The dblp.xml file has the size 820MB with 38 million data nodes.

[2] http://www.ibiblio.org/xml/books/biblegold/examples/baseball/

[3] http://www.cs.washington.edu/research/xmldatasets/

[4] http://dblp.uni-trier.de/xml/

We include the root element *dblp* in the query keyword, and include another query keyword *year* whose corresponding matches are descendant nodes of *dblp* by two levels, so the output will look like the tree depicted in Figure 10(a), where only one subtree rooted at *dblp* will be output. We pick the proper content of the *year* element to control the contribution rate. The test queries TQ17-TQ20 and the corresponding contribution rates are listed in Table 1.

From Figure 10(b), we can see that execution time of TDPrune obviously increases along with the contribution rate, while MaxMatch is not affected since it will process the same number of nodes. However, TDPrune is always more efficient than MaxMatch, and the difference is very significant.

6 Conclusion

In this paper, we propose the TDPrune system to improve the efficiency of MaxMatch. It is mainly designed to eliminate unnecessary node creations and accesses. The experimental results show that TDPrune usually outperforms MaxMatch, and the difference is sometimes by an order of magnitude. As part of our future work, we are interested in designing a novel ranking scheme to order the query results so that users may focus on the most desirable ones.

References

1. Chen, L.J., Papakonstantinou, Y.: Supporting Top-k Keyword Search in XML Databases. In: 26th International Conference on Data Engineering, pp. 689–700. IEEE Press, New York (2010)
2. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: XRANK: Ranked Keyword Search over XML Documents. In: ACM SIGMOD International Conference on Management of Data, pp. 16–27. ACM Press, New York (2003)
3. Koloniari, G., Pitoura, E.: LCA-based Selection for XML Document Collections. In: 19th International Conference on World Wide Web, pp. 511–520. ACM Press, New York (2010)
4. Li, G., Feng, J., Wang, J., Zhou, L.: Effective Keyword Search for Valuable LCAs over XML Documents. In: 16th ACM Conference on Information and Knowledge Management, pp. 31–40. ACM Press, New York (2007)
5. Li, Y., Yu, C., Jagadish, H.V.: Schema-free XQuery. In: 30th International Conference on Very Large Data Bases, pp. 72–83. Morgan Kaufmann, San Francisco (2004)
6. Liu, Z., Chen, Y.: Reasoning and Identifying Relevant Matches for XML Keyword Search. J. PVLDB 1(1), 921–932 (2008)
7. Xu, Y., Papakonstantinou, Y.: Efficient LCA Based Kyword Search in XML Data. In: 11th International Conference on Extending Database Technology, pp. 535–546. ACM Press, New York (2008)
8. Xu, Y., Papakonstantinou, Y.: Efficient Keyword Search for Smallest LCAs in XML Databases. In: ACM SIGMOD International Conference on Management of Data, pp. 537–538. ACM Press, New York (2005)
9. Zhou, R., Liu, C., Li, J.: Fast ELCA Computation for Keyword Queries on XML Data. In: 13th International Conference on Extending Database Technology, pp. 549–560. ACM Press, New York (2010)

Semi-supervised Clustering of Graph Objects: A Subgraph Mining Approach

Xin Huang[1], Hong Cheng[1], Jiong Yang[2], Jeffery Xu Yu[1],
Hongliang Fei[3], and Jun Huan[3]

[1] The Chinese University of Hong Kong
[2] Case Western Reserve University
[3] University of Kansas
{xhuang,hcheng,yu}@se.cuhk.edu.hk, jxy55@case.edu,
{hfei,jhuan}@ittc.ku.edu

Abstract. Semi-supervised clustering has recently received a lot of attention in the literature, which aims to improve the clustering performance with limited supervision. Most existing semi-supervised clustering studies assume that the data is represented in a vector space, e.g., text and relational data. When the data objects have complex structures, e.g., proteins and chemical compounds, those semi-supervised clustering methods are not directly applicable to clustering such graph objects.

In this paper, we study the problem of semi-supervised clustering of data objects which are represented as graphs. The supervision information is in the form of pairwise constraints of must-links and cannot-links. As there is no predefined feature set for the graph objects, we propose to use discriminative subgraph patterns as the features. We design an objective function which incorporates the constraints to guide the subgraph feature mining and selection process. We derive an upper bound of the objective function based on which, a branch-and-bound algorithm is proposed to speedup subgraph mining. We also introduce a redundancy measure into the feature selection process in order to reduce the redundancy in the feature set. When the graph objects are represented in the vector space of the discriminative subgraph features, we use semi-supervised kernel K-means to cluster all graph objects. Experimental results on real-world protein datasets demonstrate that the constraint information can effectively guide the feature selection and clustering process and achieve satisfactory clustering performance.

Keywords: Semi-supervised clustering, frequent subgraph mining.

1 Introduction

Complex structures in many scientific applications can be represented as graphs, e.g., protein structures, chemical compounds, program flows and XML documents. In many applications, it would be very useful if we can automatically partition a set of data objects which are represented as graphs into disjoint clusters. For example, in bioinformatics, graph clustering can distinguish different families of proteins based on their structural similarity. In practice, we may also have some prior information about the

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 197–212, 2012.
© Springer-Verlag Berlin Heidelberg 2012

graph data objects, e.g., some proteins are similar (or dissimilar) based on the similarities of their amino acid sequences and three-dimensional structure, or some proteins share a common evolutionary origin. If we can effectively incorporate the prior information into clustering, the clustering performance could be significantly boosted.

Semi-supervised clustering has recently received a lot of attention in the literature. Traditional clustering approaches fall into the category of unsupervised learning, as only unlabeled data is used for clustering. When a small amount of supervision information is available, it can be incorporated into the clustering process to improve the clustering performance. There has been research focusing on constraint-based [1] or distance-based [2], [3], [4], [5] semi-supervised clustering. However, most existing semi-supervised clustering methods assume that the input data is in a feature vector space, e.g., text and relational data. When the data objects have complex structures but no predefined feature space, such as proteins and chemical compounds, these methods are not directly applicable to cluster the data objects.

In this paper, we study the problem of clustering graph objects with a limited amount of supervision information. Supervision in the form of pairwise constraints is usually more realistic than requiring class labels in many applications. Thus we consider supervision information including *must-links* and *cannot-links*, indicating respectively whether two graph objects should belong to the same cluster or not. Such pairwise constraints occur naturally in many domains.

The first challenge in clustering graph objects is the lack of feature vector representation of the graph objects. As an effective solution adopted in recent graph classification methods [6], [7], [8], [9], we use subgraphs as features to represent a graph object in a binary vector. But different from graph classification as a supervised learning problem, we do not have class labels in our clustering problem to supervise the feature selection process. In order to evaluate the usefulness of the subgraph features, we propose a semi-supervised feature mining and selection algorithm – an objective function for subgraph feature selection is designed which incorporates the pairwise constraints, with the aim to satisfy as many constraints as possible. In order to avoid exhaustive enumeration of all subgraph features, we integrate the objective function into the subgraph mining process and push it deep for pruning the search space. Given any subgraph g, an upper bound of the objective function for g's supergraphs can be derived, based on which, we develop a branch-and-bound algorithm to efficiently search for optimal subgraph features by pruning the subgraph search space. In addition, considering the high redundancy between subgraph patterns, we design a redundancy control mechanism in order to generate a redundancy-aware feature set. Based upon the subgraphs features, all graph objects can be represented in a feature space. Then we perform the semi-supervised kernel K-means algorithm [10] to cluster the graph objects. Experimental results on real protein graphs demonstrate that our semi-supervised feature selection and graph object clustering algorithms can accurately generate clusters which are very close to the underlying family labels of the protein data. The branch-and-bound algorithm expedites the subgraph mining process and prunes a lot of low-quality subgraph features by considering both the constraint and unconstraint graph objects.

The rest of the paper is organized as follows. Section 2 discusses related work on semi-supervised clustering, graph clustering and graph mining methods. We define the semi-supervised graph clustering problem in Section 3. In Section 4 we formulate the subgraph feature mining problem as an optimization problem and develop a branch-and-bound algorithm for the feature mining. We discuss two clustering algorithms we have implemented in Section 5. Experimental results are presented in Section 6. Finally we conclude our paper in Section 7.

2 Related Work

Semi-supervised clustering algorithms aim to improve clustering results using limited supervision. The supervision is generally given as pairwise constraints. [2] proposed an algorithm that, given examples of similar or dissimilar pairs of points in $\mathbb{R}^n$, learns a distance metric over $\mathbb{R}^n$ that respects these relationships. [4] studied the problem of learning distance metrics using side-information in the form of equivalence relations, which provide small groups of data points that are known to be similar or dissimilar. [5] proposed a probabilistic model for semi-supervised clustering based on Markov Random Fields that provides a principled framework for incorporating supervision into prototype-based clustering. Most semi-supervised clustering methods in the literature assume that the input is in a vector space [1], [2], [3], [4], [5]. [10] proposed a semi-supervised clustering algorithm SS-Kernel-Kmeans, which uses a kernel approach to cluster a large graph into k disjoint components.

Most existing studies on graph clustering aim to find a k-way disjoint partitioning of a large graph to minimize a certain objective function, such as ratio cut and normalized cut [11]. Other graph clustering criteria include modularity [12], density [13], and stochastic flows [14]. To the best of our knowledge, this paper is the first work on semi-supervised feature selection and clustering of a set of graph objects.

Extracting subgraph patterns from graph data has been studied a lot. Frequent subgraph mining methods can be categorized into two major approaches: an Apriori-based approach [15], [16] and a pattern-growth approach [17], [18], [19]. Recently, graph classification [6], [7], [8] has received a lot of attention. Kong and Yu studied the semi-supervised feature selection for graph classification and proposed a solution called gSSC [9]. A common property of the above methods is to use discriminative subgraphs as the feature space for graph classification. The feature evaluation function, e.g., information gain, is integrated into the subgraph mining process. To expedite the search process, these mining algorithms may not strictly follow the traditional depth-first or breadth-first traversal order to find discriminative subgraphs, for example, [6] uses leap search to prune sibling branches, [8] follows an evolutionary computation strategy to enumerate subgraphs, and gSSC [9] uses the branch-and-bound search strategy.

3 Problem Formulation

In this section, we formulate the problem of semi-supervised clustering of graph objects based on subgraph features.

A graph object is denoted as $G = (V, E, l)$, where V is the vertex set, $E \subseteq V \times V$ is the edge set, and l is the label function mapping a vertex or an edge to a label. The size of a graph is defined as the number of edges. Given a set of graph objects

$\mathcal{D} = \{G_1, G_2, \ldots, G_n\}$ and pairwise constraints in the form of must-links and cannot-links, the goal of semi-supervised clustering is to partition the graph objects in $\mathcal{D}$ into k disjoint clusters $\{\pi_c\}_{c=1}^{k}$, where π_c represents the c-th cluster, such that the total distance between the graph objects and the corresponding cluster centroids is minimized and a minimum number of constraints are violated. The must-link constraint indicates that two graph objects should belong to the same cluster, and the cannot-link constraint indicates that two graphs should belong to different clusters. Moreover, we call the graph objects occurring in the pairwise constraints as *constraint graphs*, otherwise as *unconstraint graphs*. Thus, we can divide $\mathcal{D}$ into a constraint subset $\mathcal{D}_c$ and an unconstraint subset $\mathcal{D}_u$. $\mathcal{D} = \mathcal{D}_c \cup \mathcal{D}_u$.

Different from traditional clustering problems which assume the input is represented in a feature space, graph objects have complex topological structures, but no predefined feature space. Thus we follow the idea of subgraph-based representation, where a set of subgraphs is used as the feature set for representing the graph objects in a feature space. A graph g is a subgraph of another graph G, if there exists a subgraph isomorphism from g to G, denoted as $g \subseteq G$. G is called a supergraph of g. The definitions of subgraph isomorphism and subgraph frequency are given as follows.

Definition 1 (Subgraph Isomorphism). *For two labeled graphs g and G, a subgraph isomorphism is an injective function $f : V(g) \rightarrow V(G)$, s.t., (1) $\forall v \in V(g), l(v) = l'(f(v))$; and (2) $\forall (u, v) \in E(g), (f(u), f(v)) \in E(G)$ and $l(u, v) = l'(f(u), f(v))$, where l and l' are the labeling functions of g and G, respectively.*

Definition 2 (Frequency). *Given a graph dataset $\mathcal{D} = \{G_1, \ldots, G_n\}$ and a subgraph g, the supporting graph set of g is $\mathcal{D}_g = \{G_i | g \subseteq G_i, G_i \in \mathcal{D}\}$. The frequency of g is $\frac{|\mathcal{D}_g|}{|\mathcal{D}|}$, denoted as $freq(g)$.*

Given a set of subgraph features $\{g_1, \ldots, g_m\}$, a graph G_i can be represented as a binary vector $\mathbf{x}_i = [x_i^1, \ldots, x_i^m]^\mathsf{T}$, where the k-th component x_i^k in $\mathbf{x}_i$ denotes whether g_k is a subgraph of G_i. $x_i^k = 1$ iff $g_k \subseteq G_i$, $x_i^k = 0$ otherwise. Due to the expressiveness of subgraph features, we adopt the subgraph-based feature representation in our clustering framework. In the paper we use the following notations.

- $\mathcal{S} = \{g_1, g_2, \ldots, g_m\}$: the full set of subgraph features that can be enumerated from the graph objects in $\mathcal{D}$. Only a subset of subgraph features $T \subseteq \mathcal{S}$ is selected for graph object clustering.
- X: the matrix representation of the graph objects $\mathcal{D} = \{G_1, \ldots, G_n\}$ in the feature space of $\mathcal{S}$. $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n] = [\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_m]^\mathsf{T} \in \{0, 1\}^{m \times n}$, where $X = [X_{ij}]^{m \times n}$. $X_{ij} = 1$ iff $g_i \subseteq G_j$, $X_{ij} = 0$ otherwise.
- $\mathcal{M}_0$ and $\mathcal{C}_0$: $\mathcal{M}_0 = \{(G_i, G_j) | \pi(G_i) = \pi(G_j)\}$ denotes the given set of *must-link* constraints where a must-link indicates that two graphs should belong to the same cluster. Here $\pi(G_i)$ denotes the cluster label of graph G_i. $\mathcal{C}_0 = \{(G_i, G_j) | \pi(G_i) \neq \pi(G_j)\}$ denotes the given set of *cannot-link* constraints where a cannot-link indicates that two graphs should belong to different clusters.

Example 1. In Fig.1, we show a set of graph objects $\mathcal{D} = \{G_1, G_2, G_3\}$ and a set of subgraph features $T = \{g_1, g_2, g_3, g_4\}$. There is a must-link between (G_1, G_2) and a

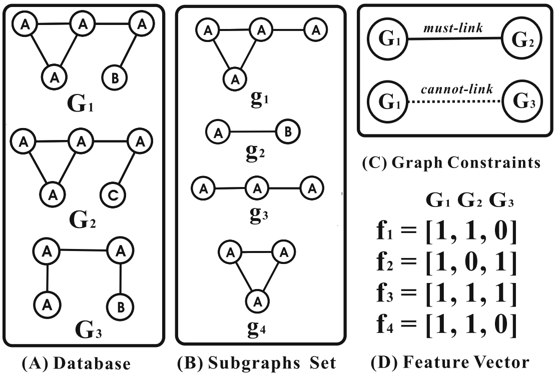


Fig. 1. A Running Example

cannot-link between (G_1, G_3). The feature vectors are also shown. For example, g_1 is a subgraph of G_1 and G_2, but not a subgraph of G_3. So the corresponding feature vector is $f_1 = [1, 1, 0]$.

4 Semi-supervised Subgraph Mining

The first and perhaps the biggest challenge in our graph object clustering problem is how to mine discriminative subgraph features based on both constraint and unconstraint graphs. It is infeasible and unnecessary to enumerate all subgraph patterns from $\mathcal{D}$ for the clustering purpose, as the number of subgraphs is exponential to the graph size. In graph classification [6], [7], [8], [9] where the class label information is available, an evaluation measure such as information gain can be used to select discriminative subgraph features. However, in our clustering problem, the limited supervision is in the form of pairwise constraints, rather than class labels. It is non-trivial to design an objective function for subgraph feature selection, with the aim to satisfy as many constraints as possible. In addition, we need a strategy to integrate the objective function into the subgraph mining process, in order to discover the set of optimal subgraph features wrt. the objective function in a timely fashion, and effectively prune the search space composed of low-quality features.

As our goal is to find a set of high-quality subgraphs for clustering wrt. the constraints, we first formulate the subgraph feature mining problem as an optimization problem, given the semi-supervised information:

$$T^* = \arg\max_{T \subseteq \mathcal{S}} \Psi(T) \ s.t. \ |T| \leq t, \tag{1}$$

where $\Psi(T)$ is an objective function to evaluate the usefulness of a subgraph feature subset T, T^* is the optimal set of subgraph features, $|T|$ represents the size of the subgraph feature set T, and t is the maximum feature number we use.

4.1 Objective Function

We consider both constraint and unconstraint graph objects in defining the objective function $\varPsi$. To fully utilize the supervision information, in the preprocessing step, we try to infer additional constraints from the given constraint sets $\mathcal{M}_0$ and $\mathcal{C}_0$ by assuming consistency of the constraints. For the must-links in $\mathcal{M}_0$, we compute the transitive closure of the must-links to derive connected components consisting of graph objects connected by must-links. Let there be κ connected components, which are used to create κ initial clusters $\{\Re_p\}_{p=1}^{\kappa}$. We use $\mathcal{M}_{inf}$ to denote the must-link constraints inferred from the transitive closure that were not in the initial set, and use $\mathcal{M} = \mathcal{M}_0 \cup \mathcal{M}_{inf}$ to denote the augmented must-link set. For each pair of initial clusters $\Re_p$ and $\Re_q$ that have at least one cannot-link between them, we add cannot-link constraints between every pair of graphs in $\Re_p$ and $\Re_q$, and denote the inferred cannot-links as $\mathcal{C}_{inf}$. The augmented cannot-link set is denoted as $\mathcal{C} = \mathcal{C}_0 \cup \mathcal{C}_{inf}$. This augmentation step can infer as many additional constraints as possible from the given constraint sets. Considering both constraint and unconstraint graphs, the objective function on subgraph features should satisfy the following aspects:

- *must-link*: each pair of graph objects $(G_i, G_j) \in \mathcal{M}_0$ should be close to each other;
- *cannot-link*: each pair of graph objects $(G_i, G_j) \in \mathcal{C}_0$ should be far away from each other;
- *separability*: unconstraint graph objects should be separated from each other. Subgraph features that are too frequent or too rare are not useful, as graph objects represented in such feature space cannot be separated from each other;
- *inner-cluster distance*: graph objects in the same initial cluster $\Re_p$ should be close to each other;
- *inter-cluster distance*: graph objects in different initial clusters $\Re_p$ and $\Re_q$ should be far away from each other.

Based on the above properties, we define an objective function $\varPsi(T)$ which we want to maximize on a subgraph feature set T as follows:

$$\begin{aligned}\varPsi(T) = {} & \frac{\alpha}{|\mathcal{C}_0|} \sum_{(G_i,G_j)\in\mathcal{C}_0} (D_T\mathbf{x}_i - D_T\mathbf{x}_j)^2 - \frac{\beta}{|\mathcal{M}_0|} \sum_{(G_i,G_j)\in\mathcal{M}_0} (D_T\mathbf{x}_i - D_T\mathbf{x}_j)^2 \\ & + \frac{\gamma}{|\mathcal{D}_u|^2} \sum_{G_i,G_j\in\mathcal{D}_u} (D_T\mathbf{x}_i - D_T\mathbf{x}_j)^2 - \frac{\delta}{|\mathcal{M}|} \sum_{p=1}^{\kappa} \sum_{\substack{G_i,G_j\in\Re_p,\\ (G_i,G_j)\in\mathcal{M}}} (D_T\mathbf{x}_i - D_T\mathbf{x}_j)^2 \\ & + \frac{\eta}{|\mathcal{C}|} \sum_{\substack{p,q=1,\\ p\neq q}}^{\kappa} \sum_{\substack{G_i\in\Re_p,G_j\in\Re_q,\\ (G_i,G_j)\in\mathcal{C}}} (D_T\mathbf{x}_i - D_T\mathbf{x}_j)^2 \qquad (2)\end{aligned}$$

where $D_T = diag(d(T))$ is a diagonal matrix indicating which features are selected into the feature set T from $\mathcal{S}$, $d(T)_i = I(g_i \in T)$ is an indicator function. $\alpha, \beta, \gamma, \delta, \eta$ are five parameters, which adjust the weights of the five types of constraints.

For two graphs $G_i, G_j \in \mathcal{D}$, we define a symmetric matrix $W = [W_{ij}]^{n\times n}$ as:

$$W_{ij} = \begin{cases} -\frac{2\beta}{|\mathcal{M}_0|} - \frac{2\delta}{|\mathcal{M}|} & \text{if } (G_i, G_j) \in \mathcal{M}_0 \\ -\frac{2\delta}{|\mathcal{M}|} & \text{if } (G_i, G_j) \in \mathcal{M}_{inf} \\ \frac{2\alpha}{|\mathcal{C}_0|} + \frac{2\eta}{|\mathcal{C}|} & \text{if } (G_i, G_j) \in \mathcal{C}_0 \\ \frac{2\eta}{|\mathcal{C}|} & \text{if } (G_i, G_j) \in \mathcal{C}_{inf} \\ \frac{2\gamma}{|\mathcal{D}_u|^2} & \text{if } G_i, G_j \in \mathcal{D}_u \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

We give a higher weight W_{ij} to the given must-links and cannot-links in $\mathcal{M}_0$ and $\mathcal{C}_0$, and a lower weight to those inferred constraints in $\mathcal{M}_{inf}$ and $\mathcal{C}_{inf}$, as we assume the provided constraints are stronger than the inferred ones. Then we can rewrite the objective function $\Psi(T)$ in Eq.(2) as follows:

$$\begin{aligned} \Psi(T) &= \frac{1}{2}\sum_{i,j}(D_T\mathbf{x}_i - D_T\mathbf{x}_j)^2 W_{ij} = trace(D_T^\top X(D-W)X^\top D_T) \\ &= trace(D_T^\top X L X^\top D_T) = \sum_{g_k\in T}(\mathbf{f}_k^\top L\mathbf{f}_k) \end{aligned}$$

where D is a diagonal matrix whose entries are column sums of W, i.e., $D_{ii} = \sum_j W_{ij}$. We denote the matrix $D - W$ as L.

When we use a feature evaluation measure q to denote $q(g_k) = \mathbf{f}_k^\top L\mathbf{f}_k$, the optimization problem in Eq.(1) can be rewritten as

$$\max_{T\subseteq\mathcal{S}} \sum_{g_k\in T} q(g_k) \; s.t. \; |T| \le t \tag{4}$$

Suppose the values for all subgraphs are denoted as $q(g_1) \ge q(g_2) \ge \ldots \ge q(g_m)$ in the descending order. The optimal solution to the optimization problem is: $T^* = \{g_i | i \le t\}$.

Example 2. Continue our example in Fig.1. After we propagate the two given constraints, we generate two initial clusters $\Re_1 = \{G_1, G_2\}$, $\Re_2 = \{G_3\}$. Suppose all five parameters $\alpha, \beta, \gamma, \delta, \eta$ are set to be 1, we can compute $W = \begin{pmatrix} 0 & -2 & \frac{3}{2} \\ -2 & 0 & \frac{1}{2} \\ \frac{3}{2} & \frac{1}{2} & 0 \end{pmatrix}$, and the corresponding matrix $L = \begin{pmatrix} -\frac{1}{2} & 2 & -\frac{3}{2} \\ 2 & -\frac{3}{2} & -\frac{1}{2} \\ -\frac{3}{2} & -\frac{1}{2} & 2 \end{pmatrix}$. With L, all subgraph features' scores can be calculated according to our objective function as: $q(g_1) = \mathbf{f}_1^\top L\mathbf{f}_1 = 2$, $q(g_2) = -\frac{3}{2}$, $q(g_3) = 0$, and $q(g_4) = 2$. g_2 has the lowest score, as it violates the must-link (G_1, G_2) and the cannot-link (G_1, G_3). g_1 and g_4 are the best features.

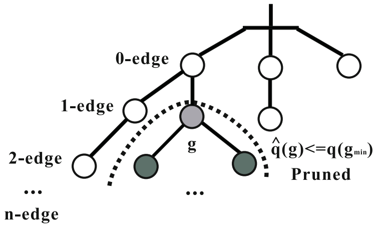


Fig. 2. Subgraph Pattern Searching Tree

4.2 Subgraph Mining with Branch-and-Bound Pruning

To select the optimal feature set T^*, we need to find t subgraphs $g_1, \ldots, g_t$ from $\mathcal{D}$ with the highest scores $q(\cdot)$. A straightforward solution is to enumerate the full set of subgraphs $\mathcal{S}$ first, and then calculate the q scores and return the top-t subgraphs. Obviously, this two-step process is not scalable, as the number of subgraphs in $\mathcal{S}$ is exponential to the size of graph objects in $\mathcal{D}$, and could be extremely large. Thus the exhaustive enumeration approach is too expensive to be practical.

Our subgraph feature mining is built based on the gSpan algorithm by Yan and Han [17]. gSpan is an efficient depth-first search algorithm to enumerate subgraphs in their minimum DFS code order. Given a minimum support threshold $min\_sup \in [0, 1]$, gSpan outputs all subgraphs whose frequency is no less than the minimum support.

To further improve the mining efficiency, we can integrate the feature evaluation function $q(\cdot)$ into gSpan and push it deep for search space pruning. If we can derive a tight upper bound of the feature evaluation function q, we can follow a branch-and-bound search strategy to quickly identify the top subgraphs and prune low-quality subgraph features. Theorem 1 gives an upper bound of the q function. The similar principle has been used in some related studies on graph mining and classification [6], [9].

Theorem 1. *(Upper Bound of q Function): Given two subgraphs $g, g' \in \mathcal{S}$, g' is a supergraph of g, i.e., $g \subseteq g'$. The q value of g', $q(g')$, is upper bounded by $\hat{q}(g)$, which is defined as : $\hat{q}(g) = \mathbf{f}_g^\top \hat{L} \mathbf{f}_g$, where the matrix $\hat{L}$ is defined as $\hat{L}_{ij} = \max(0, L_{ij})$.*

Proof. We compute $q(g') = \mathbf{f}_{g'}^\top L \mathbf{f}_{g'} = \sum_{G_i, G_j \in \mathcal{D}_{g'}} L_{ij}$ where $\mathcal{D}_{g'} = \{G_i | g' \subseteq G_i, G_i \in \mathcal{D}\}$. Since g' is the supergraph of g, we have $\mathcal{D}_{g'} \subseteq \mathcal{D}_g$ according to the Apriori property. Moreover, $\hat{L}_{ij} = \max(0, L_{ij})$, we have $\hat{L}_{ij} \geq L_{ij}$ and $\hat{L}_{ij} \geq 0$. Therefore, we have

$$q(g') = \sum_{G_i, G_j \in \mathcal{D}_{g'}} L_{ij} \leq \sum_{G_i, G_j \in \mathcal{D}_{g'}} \hat{L}_{ij} \leq \sum_{G_i, G_j \in \mathcal{D}_g} \hat{L}_{ij} = \hat{q}(g)$$

For now, we complete the proof.

With the derived upper bound $\hat{q}$, we can develop a branch-and-bound subgraph mining algorithm on top of gSpan for mining the optimal subgraph feature set T^*. During the depth-first search of the DFS code tree, we maintain the current top-t subgraphs according to the q function. Let g be the currently visited subgraph in the DFS code tree. If there are less than t subgraphs in T, we directly add g into T and recursively perform mining in the DFS code tree; if there are t subgraphs in T, then we will check whether the q value of g, $q(g)$ is higher than the current minimum q value in T, i.e., $q(g) > \min_{g' \in T} q(g')$. If yes, we will replace the lowest-ranked subgraph in T, i.e., $\arg\min_{g' \in T} q(g')$, with g. Before we recursively search the subtree rooted at g, we will first estimate the upper bound of any supergraph g' of g by $\hat{q}(g) = \mathbf{f}_g^\top \hat{L} \mathbf{f}_g$. If $\hat{q}(g)$ is less than the minimum q value in T we have so far, we can safely prune the DSF code subtree rooted at g, as all supergraphs of g cannot have a higher q value than the current top-t subgraphs in T. Fig. 2 illustrates the idea of branch-and-bound search in the DFS code tree. Algorithm 1 shows the branch-and-bound algorithm.

Algorithm 1. Branch-and-Bound Subgraph Mining

Input: Graph objects $\mathcal{D} = \{G_1, \ldots, G_n\}$, must-links $\mathcal{M}$ and cannot-links $\mathcal{C}$, minimum support threshold min_sup, maximum number of features selected t

Output: A set of optimal subgraph features T^*

```
formulate the subgraph feature evaluation function $q$ from $\mathcal{M}$ and $\mathcal{C}$;
$T \leftarrow \emptyset$;
recursively DFS traverse the DFS Code Tree in gSpan:
    $g \leftarrow$ currently visited subgraph in DFS Code Tree;
    if $|T| < t$
        $T \leftarrow T \cup \{g\}$;
        recursively DFS traverse the subtree rooted at $g$;
    else if $q(g) > \min_{g' \in T} q(g')$
        $g_{min} = \arg\min_{g' \in T} q(g')$ and $T = T - \{g_{min}\}$;
        $T = T \cup \{g\}$ and update $g_{min} = \arg\min_{g' \in T} q(g')$;
    if $\hat{q}(g) > q(g_{min})$ and $freq(g) \geq min\_sup$
        recursively DFS traverse the subtree rooted at $g$;
return $T^* = T$
```

4.3 Redundancy-Aware Subgraph Features

Based on the feature evaluation function q, we aim to find t subgraph features with the highest q function scores. However, there is a potential issue due to the high redundancy between subgraph patterns: a graph pattern g often occurs in a similar set of graph objects in the database with its supergraph or subgraph patterns. If a graph has a very high q function score, it is likely that its supergraphs or subgraphs have high scores as well. But such supergraphs and subgraphs are redundant to each other. If we return t subgraph features with high redundancy, the useful information contained in the t features is not maximized. To avoid this case, we introduce another function R to measure the redundancy between two subgraphs by the overlap of their supporting graph sets. Given two subgraphs g_i and g_j, the redundancy is defined as $R(g_i, g_j) = \frac{|\mathcal{D}_{g_i} \cap \mathcal{D}_{g_j}|}{|\mathcal{D}_{g_i} \cup \mathcal{D}_{g_j}|}$, where $\mathcal{D}_g$ is the set of graph objects containing a subgraph g. $R \in [0, 1]$. It measures the co-occurrences of two subgraphs in the graph database. The higher $R(g_i, g_j)$ is, the more redundant g_i and g_j are.

Taking the redundancy between graphs into consideration, our goal is to find t subgraphs which have high q function scores and low mutual redundancy. Formally we set a redundancy threshold $\delta \in [0, 1]$. For any two graphs g_i, g_j in the answer set T, we require $R(g_i, g_j) \leq \delta$. With the redundancy requirement, our branch-and-bound subgraph mining algorithm (Algorithm 1) can be revised as follows. Let g be the currently visited subgraph pattern. If $q(g) > \min_{g' \in T} q(g')$, we further check the redundancy between g and every graph $g' \in T$. If $\forall g' \in T$ where $q(g') \geq q(g)$, we have $R(g, g') \leq \delta$ hold, then g is added to T. Otherwise, g is not added to T and we proceed with the recursive subgraph mining process. If g is added to T, then we further check for every $g' \in T$ where $q(g') < q(g)$. If $R(g, g') > \delta$, we will remove g' from T, to make sure every pair of subgraphs in T satisfy the redundancy requirement. Our revised algorithm makes a tradeoff between the feature optimality (wrt. the q function) and the redundancy. For a high-quality subgraph feature with a high q value, if it is redundant to another good feature in the answer set, then the former one will not be considered. With such redundancy control, we will select a high-quality feature set with diversity.

Example 3. In Fig. 1, g_1 is a supergraph of g_4. As both g_1 and g_4 are subgraphs of G_1 and G_2, we have $\mathcal{D}_{g_1} = \mathcal{D}_{g_4} = \{G_1, G_2\}$. The redundancy $R(g_1, g_4) = \frac{|\{G_1, G_2\} \cap \{G_1, G_2\}|}{|\{G_1, G_2\} \cup \{G_1, G_2\}|} = 1$. With the redundancy control mechanism, we choose only one of them.

5 Semi-supervised Graph Object Clustering

Based on the optimal subgraph feature set T^*, we can represent each graph object $G_i \in \mathcal{D}$ as a vector $\mathbf{x}_i$. Then we can use traditional clustering approaches to cluster the vector representation of the graphs objects. In this section, we describe two clustering algorithms we have tested. One is the widely used clustering algorithm K-means, and the other is the kernel-based semi-supervised clustering algorithm SS-Kernel-Kmeans [10]. We use squared Euclidean distance as the unified clustering distortion measure.

5.1 K-Means

In K-means, we first choose k random points as initial centroids. Then each point is assigned to the closest centroid. After that, the centroid of each cluster is updated by taking the average of the vectors of all points in that cluster. We repeat the point assignment and centroid update steps until no point changes its cluster assignment, or we reach the user-specified maximum iterations. The goal of K-means clustering is to find k clusters $\{\pi_c\}_{c=1}^k$ which minimize the sum of the squared distance of each point to its closest centroid. The objective function we aim to minimize can be expressed as:

$$\mathcal{J}\left(\{\pi_c\}_{c=1}^k\right) = \sum_{c=1}^{k} \sum_{\mathbf{x}_i \in \pi_c} \|\mathbf{x}_i - \mathbf{m}_c\|^2, \text{ where } \mathbf{m}_c = \frac{\sum_{\mathbf{x}_\mathrm{i} \in \pi_\mathrm{c}} \mathbf{x}_i}{|\pi_c|} \tag{5}$$

Note that in K-means, we do not utilize the must-links $\mathcal{M}$ and cannot-links $\mathcal{C}$.

5.2 Semi-supervised Kernel-Kmeans

In this part, we discuss the semi-supervised kernel K-means algorithm [10] which considers must-link and cannot-link constraints during the clustering process. Assume two points $\mathbf{x}_i$, $\mathbf{x}_j$ belong to clusters π_p, π_q respectively. The objective function we aim to minimize can be formulated as:

$$\mathcal{J}\left(\{\pi_c\}_{c=1}^{k}\right) = \sum_{c=1}^{k} \sum_{\mathbf{x}_i \in \pi_c} \|\mathbf{x}_i - \mathbf{m}_c\|^2 - \sum_{\substack{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{M} \\ \pi_p=\pi_q}} \frac{\hat{w}_{ij}}{|\pi_p|} + \sum_{\substack{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{C} \\ \pi_p=\pi_q}} \frac{\hat{w}_{ij}}{|\pi_p|} \quad (6)$$

The first term in Eq.(6) is the standard K-means objective function, the second term is a cluster-size weighted reward function for must-link constraint satisfaction, and the third is a cluster-size weighted penalty function for cannot-link constraint violation.

6 Experimental Study

In this section, we report our experimental results to demonstrate the effectiveness and mining efficiency of our semi-supervised feature selection and clustering methods. Our algorithm is implemented in C++ and compiled with g++ 2.95.3. The experiments are preformed on a machine with 2.66GHz CPU.

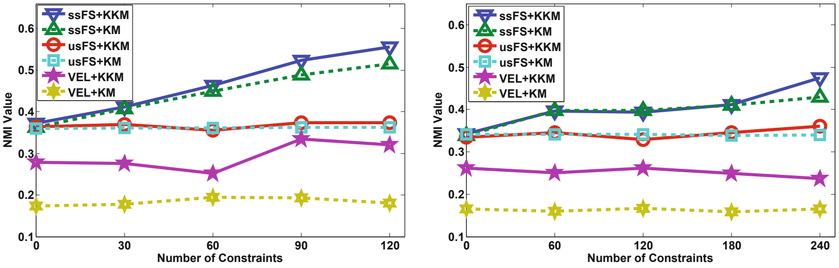
6.1 Datasets

We use protein datasets in our experiments. The protein datasets consist of protein structures from Protein Data Bank (http://www.rcsb.org/pdb/) classified by SCOP (Structural Classification of Proteins). A protein can be represented as a graph object, where a node represents an amino acid and is labeled with the amino acid type. An edge exists between two nodes if the distance between the two alpha carbons in the amino acids is less than 11.5 angstroms and the edge is labeled based on the distance between the alpha carbons. The protein families we use and their sizes are listed in Table 1. The average node number is 217 and the average edge number is 2141 in a protein graph. In the first group of experiments, the graph dataset is created by selecting three protein families with SCOP id 52592, 56251, and 56437. We set the cluster number $k = 3$ in this experiment. In the second group of experiments, the dataset consists of all six protein families listed in Table 1. We set $k = 6$ in this experiment.

Table 1. List of SCOP Families

SCOP ID	Family Name	Number of Proteins
47617	Glutathione S-transferase (GST)	36
50514	Eukaryotic proteases	44
52592	G proteins	33
56251	Proteasome subunits	35
56437	C-type lection domains	38
88634	Picornaviridae-like VP	39

To generate the pairwise constraints, we randomly select pairs of graph objects (G_i, G_j) from the protein dataset. If G_i and G_j belong to the same family, we create a must-link between G_i and G_j; otherwise, we create a cannot-link between them.



(a) Cluster Number $k = 3$

(b) Cluster Number $k = 6$

Fig. 3. Clustering Performance of Different Methods

6.2 Clustering Methodology and Evaluation Measure

We perform 2-fold cross validation in our experiments: 50% of the graph dataset is used as the training set, from which the must-link and cannot-link constraints are sampled. The other 50% of the dataset is used as the test set. The subgraph mining and clustering steps are run on the whole dataset, while the clustering evaluation is done on the test set only. All results are averaged over 10 runs of the 2-fold cross validation. In the feature set objective function Ψ in Eq.(2), all five parameters $\alpha, \beta, \gamma, \delta, \eta$ are set to be 1.

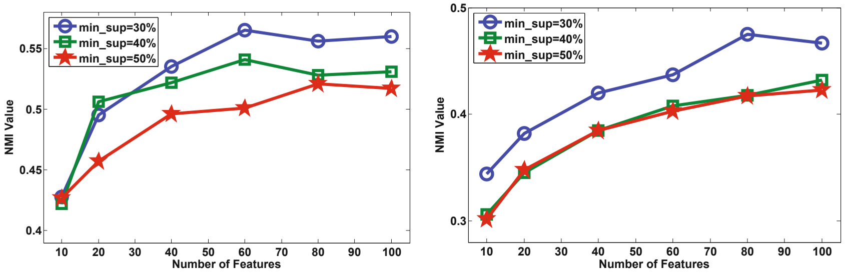
We use *normalized mutual information* (NMI) to evaluate our clustering results. It estimates how closely the clustering algorithm could reconstruct the underlying label distribution in the data. If X is the random variable denoting the cluster labels of the graph objects and Y is the random variable denoting the underlying class labels of the graphs, then NMI is defined as $NMI = \frac{I(X;Y)}{(H(X)+H(Y))/2}$, where $I(X;Y) = H(X) - H(X|Y)$ is the mutual information between the random variables X and Y, $H(X)$ is the Shannon entropy of X, and $H(X|Y)$ is the conditional entropy of X given Y.

We test the following feature selection methods, where the first two use subgraph features and the third uses simple vertex and edge based features.

- *Semi-supervised subgraph feature selection*. We select the optimal subgraph feature set T^* wrt. the objective function $\Psi(T)$ which incorporates the supervision information. This method is denoted as **ssFS**.
- *Unsupervised subgraph feature selection*. We select a feature set T wrt. the objective function $\Psi(T)$, but the constraints are not used. Thus we only consider the *separability* of the subgraph features on the unconstraint graph objects. This method is denoted as **usFS**.
- *Using vertex and edge labels as features*. As a baseline method, we use vertex and edge labels as features to represent a graph object as a binary vector. This method is denoted as **VEL**.

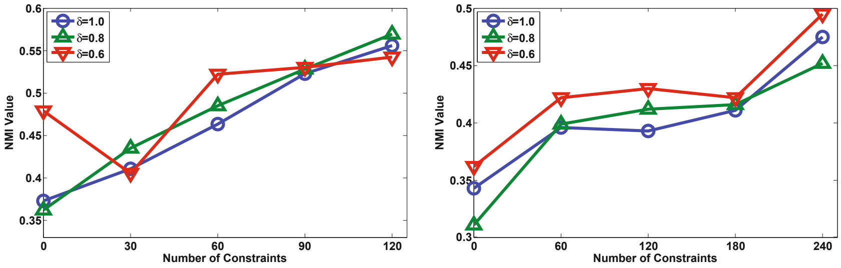
We also test the following two clustering methods.

- *Semi-supervised clustering*. The semi-supervised kernel based clustering algorithm SS-Kernel-Kmeans [10] is used for clustering. This method is denoted as **KKM**.



(a) Cluster Number $k = 3$ (b) Cluster Number $k = 6$

Fig. 4. Clustering Performance with Different Number of Features and Minimum Support



(a) Cluster Number $k = 3$ (b) Cluster Number $k = 6$

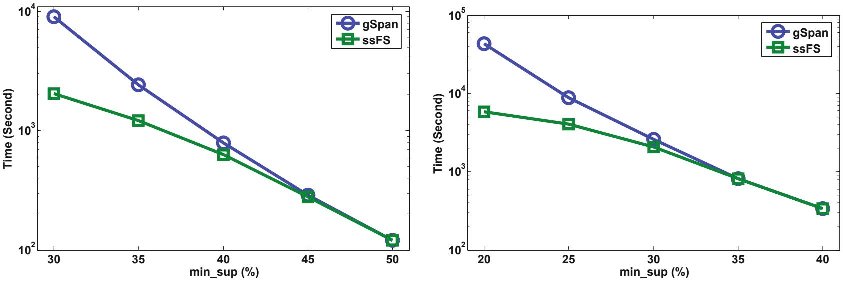
Fig. 5. Clustering Performance with Different Redundancy Thresholds

– *Unsupervised clustering*. Traditional K-means is used for clustering. This method is denoted as **KM**.

We combine different feature selection mechanisms with the two clustering methods, and compare their clustering performance.

6.3 Performance on Graph Clustering

In the first experiment, we compare the clustering performance of different methods listed above. Figure 3 shows the NMI value of different clustering methods when we increase the number of constraints on the two protein graph datasets we created ($k = 3$ and $k = 6$). We set $min\_sup = 30\%$, the number of features $t = 80$, and the redundancy threshold $\delta = 1.0$. A general trend we observe is an increasing NMI value with the increasing number of constraints. In addition, ssFS+KKM achieves the highest NMI value as it utilizes the supervision information in both feature selection and clustering. ssFS+KM comes next, which also shows the usefulness of the optimal feature set when considering constraints for feature selection. The performance of usFS+KKM and usFS+KM is much worse, as the feature selection step is unsupervised. We can see the NMI value of usFS+KM remains unchanged when we increase the constraint number, as it does not utilize the supervision information at all. Finally, the performance of



(a) Cluster Number $k = 3$ (b) Cluster Number $k = 6$

Fig. 6. Semi-supervised Subgraph Feature Mining Time with Branch-and-Bound Search

VEL is the worst, which shows that subgraph features are more effective than simple label features. This experiment demonstrates that our feature selection objective function $\Psi(T)$ is very effective in selecting discriminative subgraph features for clustering. When considering supervision information, the semi-supervised clustering algorithm can also improve the clustering performance.

In the second experiment, we test the clustering performance by varying the parameters min_sup and the feature number t. We fix the redundancy threshold $\delta = 1.0$ and use the semi-supervised method ssFS+KKM. We use 120 constraints for the 3-cluster dataset ($k = 3$) and 240 constraints for the 6-cluster dataset ($k = 6$). As we can see from Figure 4, the NMI value increases in general with the increasing number of features. The NMI value also increases when min_sup decreases, as a lower min_sup implies a larger set of subgraph candidates for feature selection.

In the third experiment, we test the clustering performance by varying the redundancy threshold δ. We set $min\_sup = 30\%$ and the number of features $t = 80$. We run the semi-supervised method ssFS+KKM. As we can see from Figure 5, in general the clustering performance improves as the redundancy threshold decreases, i.e., with a lower redundancy tolerance. In most cases, the NMI value is the highest when $\delta = 0.6$ and it is the lowest when $\delta = 1.0$, i.e., with no redundancy control. For all three redundancy thresholds, the NMI values increase with the number of available constraints.

6.4 Subgraph Mining Efficiency

In this part, we study the subgraph mining efficiency of the branch-and-bound search algorithm (Algorithm 1). In this experiment we set the number of features $t = 100$ and the redundancy threshold $\delta = 1.0$. For semi-supervised feature selection, we use 120 constraints for the 3-cluster dataset ($k = 3$) and 240 constraints for the 6-cluster dataset ($k = 6$). Figure 6 shows the subgraph feature mining time in a logarithmic scale by our feature selection method (ssFS) and the original gSpan mining algorithm, when we vary the min_sup threshold. When the min_sup is low, ssFS is about an order of magnitude faster than gSpan, which shows the effectiveness of the branch-and-bound based pruning. When the min_sup is high, the difference becomes smaller, as a lot of subgraph candidates can be pruned in gSpan as well simply based on min_sup.

7 Conclusions

In this paper, we study the problem of semi-supervised graph object clustering, where pairwise constraints as must-links and cannot-links are used to guide the feature selection and clustering steps. As graph objects are not represented in a vector space, we propose to use subgraphs as features to represent the graph objects in a feature space. An objective function for feature selection is designed, which incorporates the pairwise constraints. We integrate the objective function into gSpan for mining the optimal feature set. An upper bound of the objective function enables us to prune the subgraph search space effectively in a branch-and-bound manner. A semi-supervised kernel based clustering algorithm is used to cluster the graph objects. Our experiments demonstrate that the semi-supervised subgraph feature selection and clustering approach is very effective in boosting the clustering performance.

Acknowledgment. The work described in this paper was partially supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No.: CUHK 419008, 411310 and 411211). Hongliang Fei and Jun Huan are partially supported by the US NSF grant IIS 0845951.

References

1. Wagstaff, K., Cardie, C., Rogers, S., Schroedl, S.: Constrained k-means clustering with background knowledge. In: ICML, Williamstown, MA, pp. 577–584 (June 2001)
2. Xing, E.P., Ng, A.Y., Jordan, M.I., Russell, S.: Distance metric learning, with application to clustering with side-information. In: NIPS, Vancouver, BC, pp. 505–512 (December 2002)
3. Klein, D., Kamvar, S., Manning, C.: From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In: ICML, Sydney, Australia, pp. 307–314 (July 2002)
4. Bar-Hillel, A., Hertz, T., Shental, N., Weinshall, D.: Learning distance function using equivalence relations. In: ICML, Washington, DC, pp. 11–18 (August 2003)
5. Basu, S., Bilenko, M., Mooney, R.J.: A probabilistic framework for semi-supervised clustering. In: KDD, Seattle, WA, pp. 59–68 (August 2004)
6. Yan, X., Cheng, H., Han, J., Yu, P.S.: Mining significant graph patterns by scalable leap search. In: SIGMOD, Vancouver, Canada, pp. 433–444 (June 2008)
7. Ranu, S., Singh, A.K.: GraphSig: A scalable approach to mining significant subgraphs in large graph databases. In: ICDE, Shanghai, China, pp. 844–855 (March 2009)
8. Jin, N., Young, C., Wang, W.: GAIA: graph classification using evolutionary computation. In: SIGMOD, Indianapolis, IN, pp. 879–890 (June 2010)
9. Kong, X., Yu, P.S.: Semi-supervised feature selection for graph classification. In: KDD, Washington, DC, pp. 793–802 (July 2010)
10. Kulis, B., Basu, S., Dhillon, I., Mooney, R.: Semi-supervised graph clustering: A kernel approach. In: ICML, Bonn, Germany, pp. 457–464 (August 2005)
11. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Trans. Pattern Analysis and Machine Intelligence 22(8), 888–905 (2000)
12. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E 69, 026113 (2004)
13. Xu, X., Yuruk, N., Feng, Z., Schweiger, T.A.J.: SCAN: A structural clustering algorithm for networks. In: KDD, San Jose, CA, pp. 824–833 (August 2007)

14. Satuluri, V., Parthasarathy, S.: Scalable graph clustering using stochastic flows: Applications to community discovery. In: KDD, Paris, France, pp. 737–746 (June 2009)
15. Inokuchi, A., Washio, T., Motoda, H.: An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. In: Żytkow, J.M. (ed.) PKDD 1998. LNCS, vol. 1510, pp. 13–23. Springer, Heidelberg (1998)
16. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: ICDM, San Jose, CA, pp. 313–320 (November 2001)
17. Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In: ICDM, Maebashi, Japan, pp. 721–724 (December 2002)
18. Huan, J., Wang, W., Prins, J.: Efficient mining of frequent subgraph in the presence of isomorphism. In: ICDM, Melbourne, FL, pp. 549–552 (November 2003)
19. Nijssen, S., Kok, J.: A quickstart in frequent structure mining can make a difference. In: KDD, Seattle, WA, pp. 647–652 (August 2004)

Plink-LDA: Using Link as Prior Information in Topic Modeling

Huan Xia[1], Juanzi Li[1], Jie Tang[1], and Marie-Francine Moens[2]

[1] Tsinghua National Laboratory for Information Science and Technology,
Department of Computer Science and Technology, Tsinghua University,
Beijing 100084, China
[2] Department of Computer Science, Katholieke Universiteit Leuven,
Celestijnenlaan 200 A B-3001 Heverlee, Belgium
{xiahuan,ljz,tangjie}@keg.cs.tsinghua.edu.cn,
sien.moens@cs.kuleuven.be

Abstract. Citations are highly valuable for analyzing documents and have been widely studied in recent years. Among the document modeling, the citations are treated as documents' attributes just like the words in the documents; or as the degrees in graph theory. These methods add citations into word sampling process to reform the document representation but they miss the impact of the citations in the generation of content. In this paper, we view the citations as the prior information which authors have had. In the generation of document, content of the document is split into two parts: the idea of the author and the knowledge from the cited papers. We proposed a prior information enabled topic model-PLDA. In the modeling, both the document and its citations play the important role of generating the topic layer. Our experiments on two linked datasets show that our model greatly outperforms basic LDA procedures on a clustering task while also maintaining the dependencies among documents. In addition, we also show the feasibility by the task of citation recommendation.

Keywords: Topic Modeling, LDA, Links, Prior Information, Plink-LDA.

1 Introduction

In recent years, social network such as Facebook, Twitter are growing rapidly. One important and essential part in these networks is the following relationship. The following relationship plays an important role in user generated contents, for users are strongly influenced by the posts which they follow. It is the posts which produce these comments. If there were no posts, such as "#911" shown in Figure 1, there would have been no comments about these events. Similarly, authors make a reference to other documents while they are writing a paper. People cite papers for they have gained some knowledge from the existed knowledge in the literature as shown in Figure 2. Both following relationships and references indicate not only topical similarity but also dependencies between different documents. Recent studies on how to use these links can be classified into two categories: one is that links are used as document attributes just like word appearances in the documents; the other is that links are used as degrees in graph theory. However, both of them cannot handle the

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 213–227, 2012.

© Springer-Verlag Berlin Heidelberg 2012

problem of taking these two features, topical similarity and dependencies, into account simultaneously.

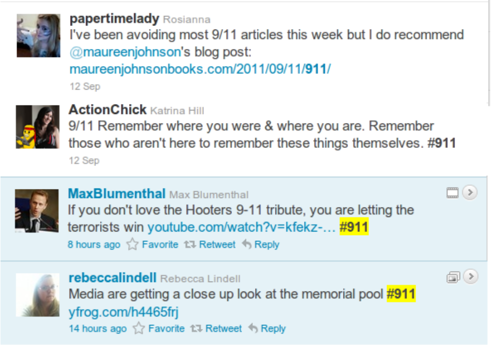
Topic model is a popular strategy to analyze the texts in the documents. Many derivations of topic models have been proposed to meet different requirements on the basis of LDA [1]. Among these models, researchers address the problem of how to integrate the links information into the model [2, 3, 4, 5, 6, 14]. However, links information is treated as attributes of documents in these models. As a result, the dependencies between documents are ignored. We think that both comments and references should be referred to because they have the influence on the generating of the content, so can we use the link information in another way to show both features? Usually, authors make a reference to other documents because what they talk about is closely related to what they cite, whether agreement or disagreement. So can we treat these links information as some kind of prior information to generate the document content? Driven by this, we address the problem of analyzing and using links in a different way. We hope that our new model can explicitly model the citations and words simultaneously and maintain both the topic similarity and dependencies, which makes our contribution in this paper:

1) We use link information which indicates the topical similarity and dependencies between documents as a kind of prior information in the generation of the document.

2) We propose a unified topic model which can model links and word simultaneously to address above problem.

3) We implement three experiments to evaluate the feasibility of our proposed model.

The rest of the paper is organized as follows. We give the formal description of our intuition in section 2. We describe our proposed model Plink-LDA by treating citations or following relationships as prior information in this paper. Plink-LDA can model the citations and content simultaneously. In order to evaluate the feasibility of our model, we do the experiments in section 4 and the experiment results show that our model outperforms the baseline. We discuss related work in section 5. Section 6 concludes the paper.



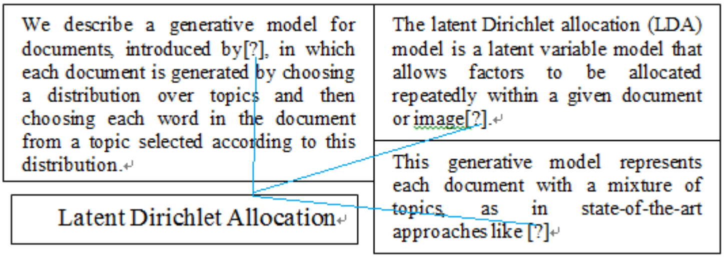
papertimelady Rosianna
I've been avoiding most 9/11 articles this week but I do recommend @maureenjohnson's blog post: maureenjohnsonbooks.com/2011/09/11/**911**/
12 Sep

ActionChick Katrina Hill
9/11 Remember where you were & where you are. Remember those who aren't here to remember these things themselves. **#911**
12 Sep

MaxBlumenthal Max Blumenthal
If you don't love the Hooters 9-11 tribute, you are letting the terrorists win youtube.com/watch?v=kfekz-... **#911**
8 hours ago ☆ Favorite Retweet Reply

rebeccalindell Rebecca Lindell
Media are getting a close up look at the memorial pool **#911** yfrog.com/h4465frj
14 hours ago ☆ Favorite Retweet Reply

Fig. 1. User generated contents example. User comments on the event 911 ten year anniversaries extracted from twitter. Users express their views on this event. The views can be seen as exactly the reactions to the post #911.



We describe a generative model for documents, introduced by[?], in which each document is generated by choosing a distribution over topics and then choosing each word in the document from a topic selected according to this distribution.

The latent Dirichlet allocation (LDA) model is a latent variable model that allows factors to be allocated repeatedly within a given document or image[?].

This generative model represents each document with a mixture of topics, as in state-of-the-art approaches like [?]

Fig. 2. Citations example. Contexts of papers which cite paper "Latent Dirichlet Allocation". These contexts are mostly introduction to or comparison with LDA. And they are highly related to each other for that they all cite the same paper.

2 Problem Definition

Latent Dirichlet Allocation (LDA) is a probabilistic model to analyze documents in the latent topic layer. It interprets the document generation as word sampling process from topics. For each word w_{di} in document d, a specific topic z_{d_i} is chosen from the document-specific distribution. Then w_{di} is generated according to the topic-specific multinomial distribution $\Phi_{z_{di}}$.

The generating process of LDA is very intuitive: first, the authors choose a topic, which concept the author wants to talk about; second, the authors choose the mostly used words for this topic to constitute the content of the document. However, this process models the word appearance only. In order to analyze the influence of the link, many derivations of LDA are proposed to model links with words simultaneously. All of these models have an underlying common view that integrating citation into model can make document distribution more precisely; for citations reveal some content that is not mentioned by words in the documents. Citations can make up this lost information which cannot be completely represented by words. Therefore, these models tend to place weight on some topics which not mentioned by the document but by its citations. So the documents will be fully represented. In these models, links are just like a special kind of word in the document and they are also generated from the document-specific distribution. Similarly, we formalized our idea in the topic pattern**:**

The author gets information z_i from a reference d_i

And he may have his own idea z after he gets many z_i

According to the mixture of z and z_i, all of the words of the document the author going to write are sampled to the topic distribution.

In this model, links are no longer sampled as results. Instead, they are the prior information of the documents. The citations make a change to the document-specific distribution which eventually reflects our idea. Before giving our model, we give some notations we are going to use in the following sections.

Definition 2.1. [Document]. The content of a document excluding the references. We use d_i to represent the i*th* document in the dataset.

Definition 2.2. [Citation (Link)]. The document's references. If document d cites another document d_i, we call d_i as a citation of document d. d is also noted as citing document and d_i as cited document. Both follow relationship on social network and references of documents are noted as citations here. We use c_i to represent the ith citation of a document.

Definition 2.3. [Related Documents]. Those documents which are talking about the same topic. Most parts of them are similar. The differences among them may only take a small part of their contents.

3 Methodology

3.1 1Intuition of Plink-LDA

To illustrate our model, we first look into some details of LDA model. LDA defines the following generating process for every document in a corpus [1]:

1. For each document d, draw a topic distribution $\theta \sim \mathrm{Dir}(\alpha)$;
2. For each word w_i in document d:
   a) Draw a topic $z_n \sim Multinomial(\theta)$.
   b) Sample a word w_i according to the multinomial condition probability distribution $\mathrm{p}(w_i|z_n, \beta)$.

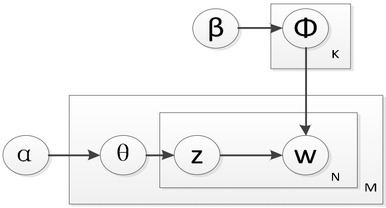


Fig. 3. Plate notation for LDA

α, β are hyper parameters for the specific collection. The probability of generating a word from a document d is:

$$\mathrm{P}(\mathrm{w}|\mathrm{d}, \theta, \Phi) = \sum_{z \in T} P(w|\, z, \Phi_z) P(z|d, \theta_d) \tag{1}$$

LDA model analyzes documents in three layers: word layer, topic layer and document layer. An informal interpretation is that: Documents generate topics and topics generate words. From the graphical representation, we can see that generating of topic layer is controlled only by the document which it belongs to.

Many derivations [4, 5, 7] of LDA integrate citations into the model during the word sampling process and topic layer is still controlled by the document itself in these models. Adding citations into word sampling process do reform the document representation; however, sometimes these reformed distributions may not produce the expected results. For example, two documents may have same citation list but they talk about totally different solutions to one problem. So some parts of the two documents will be totally different from each other but they may just take a small place of the document content. Although it is these parts which distinguish them, their small content occupation may be ignored by their great similarity in citations.

To express this intuition more clearly, we can split document content into two parts: 1) the idea of the authors; 2) the knowledge learned in the existed literature. But it is not equally reflected in the words of the documents. All of the words are supposed to be related to the first part mentioned above in previous topic models. Citations which the authors refer to are not taken into account. They are just treated as attributes of documents just like word appearances. Actually, many words in the documents are generated by the topic of the citations. In order to reveal this, we assume that the generation for words should be controlled by both the document and its citations, and also the relation should be reflected in the model.

To reflect the intuition that citations have impact on the content constitution of the documents, we propose a model which utilizes citations as prior information. To reveal this change in utilizing citations, we modify the topic sampling process on the basis of LDA. The topic sampling is controlled by both document and its citations. We combine document and its citations' topic distributions together to generate the topic. So the generating of the topic layer is no longer controlled only by the document topic distribution only. Instead, both the document and its citations' document layer play the role of generating the topic layer.

3.2 Our Model

Based on the discussion in section 3.1, we propose the model to address the problem that using citations as prior information in LDA. First we give the plate notation graph and notations in the following:

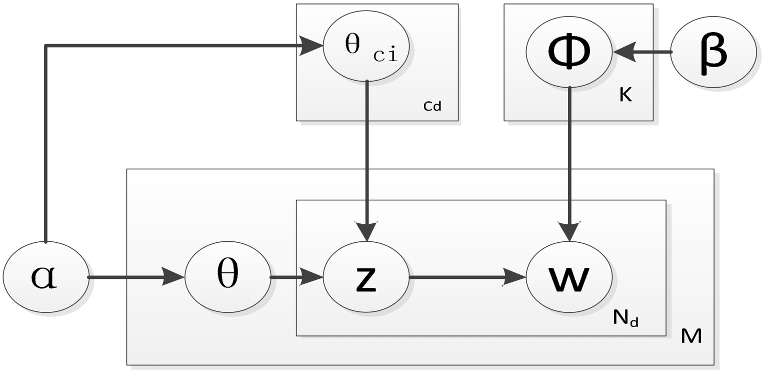


Fig. 4. Plate notation for Plink-LDA

For this model, the generative process is as follows:

1. For each document d, draw a document specific distribution θ.
2. For each word w_{di} in d:
   a) Randomly sample a citation inference c_i, then draw a document specific distribution θ_{c_i};
   b) Combine θ and θ_{c_i} by tuning parameter λ to generate a document distribution θ_c;
   c) Sample a topic z_i according to the combined topic distribution θ_c;
   d) Generate word w_{di} according to $\mathrm{P}(w_{di}|z_i, \Phi)$.

As dedicated in Figure 4, topic sampling process has taken the citation into account. To show the influence, we make a linear combination of the document and its citations' topic distribution θ controlled by an tuning parameterλ. For the generating process of 2.c, the combined topic distribution is :

$$\theta_c = (1-\lambda)\theta + \lambda\theta_{c_i} \tag{2}$$

As to sampling a citation inference c_i, we take all the citations into account for the ease and adjust the weights by the parameter λ.

To estimate the parameters for this model, we take the widely used Gibbs sampling procedure to estimate the latent variable. We use the same sampling algorithm as that for LDA model with the posterior probability:

$$\mathrm{P}\left(z_{d_i}|z_{d_{-i}}, w, c, \alpha, \beta, \lambda\right) \propto \frac{(1-\lambda n_{z_{d_i}}^{-d_i}) + \lambda n_{c_i} + \alpha_{z_{d_i}}}{\sum_z((1-\lambda)n_{dz} + \lambda n_{cz} + \alpha)} \times \frac{n_{w_i}^{-d_i} + \beta}{\sum_\mathrm{i}\left(n_{w_i} + K \cdot \beta\right)} \tag{3}$$

where "-" indicates excluding that instance from counting. The notation is as follows:

Table 1. Notations for our model

Symbol	Description
z_{d_i}	topic i assigned to word w
$z_{d_{-i}}$	topic i assigned to word w excluding current instance
w	current word
c	citations of the document
$n_{z_{d_i}}^{-d_i}$	number of words assigned to topic i in document d excluding instance of word i
n_{dz}	number of words assigned to topic z in document d
n_{c_i}	number of words assigned to topic i in citations of document d
λ	the tuning parameter
$\alpha_{z_{d_i}}$	hyper parameter for each document, $\sum \alpha_{z_{d_i}} = \alpha$
α, β	hyper parameter for LDA model
$n_{w_i}^{-d_i}$	number of words assigned to topic i of all instances of word w excluding this instance
n_{w_i}	number of words assigned to topic i
k	number of word tokens

We notice that the difference in the posterior probability between LDA and our model is whether the instances of citations are counted. The instances of words in citations actually reveal its topic representation. In our model, those words strongly related to the topics of the citations are mainly generated by the citations topic distribution. The document topic distribution is modified to show the difference between its citations and itself. This change in topic distribution is supposed to discriminate the small difference between documents when most part contents of documents are similar. The similar dimensions caused by citations in topic space are removed or slightly reduced. The modified topic distribution in our model mainly focuses on the different parts of its content. As a result, it is capable of distinguishing those documents which are strongly related.

4 Experimental Design

4.1 Datasets

For our experiments, we used two standard linked data sets: Citeseer[1] and Cora[2], to evaluate our model.

Citeseer consists of 3312 scientific publications from six categories: Agents, Artificial Intelligence, Database, Human Computer Interaction, Machine Learning and Information Retrieval. The citation network consists of 4732 links. After stemming and removing stop words, 3703 unique words remain.

Cora is a dataset containing machine learning papers published in the conferences and journals of seven categories: Neural Networks, Rule Learning, Reinforcement Learning, Probabilistic Methods, Theory, Genetic Algorithms and Case Based. For each paper, there is a unique label to indicate which category it belongs to. The Cora dataset subset consists of 2708 scientific publications classified to one of seven classes. There are 5429 citations in the data set. After preprocessing, 1433 unique words remain.

4.2 Tasks and Evaluation

4.2.1 Clustering Performance

In this task, we measure how well our model performs after integrating links as prior information into document modeling. We do the clustering task and compare the results based on our model with LDA in terms of accuracy and recall number.

The experimental set-up is as follows. We first train Latent Dirichlet Allocation model on Citeseer and Cora datasets respectively. We use these model parameter results as our baseline. Then we model these two datasets based on our proposed model iteratively. To observe the impact of tuning parameter, we model the datasets with different tuning values for λ. After this, we utilize the model parameters to automatically cluster the documents in the two datasets. After clustering, we first decide which category these clusters belong to and then we define the accuracy for a cluster as follows:

[1] `http://www.cs.umd.edu/~sen/lbc-proj/data/citeseer.tgz`

[2] `http://www.cs.umd.edu/~sen/lbc-proj/data/cora.tgz`

$$cluster\ accuracy = \frac{number\ of\ documents\ belonging\ to\ the\ cluster\ category}{number\ of\ the\ cluster\ \ dcouments} \quad (4)$$

Then we calculate the accuracy for a dataset by combining cluster accuracies with their weights together.

Figure 5 and Figure 6 show the results of clustering on two linked datasets. For Cora and Citeseer, the model parameters, topic number, are set to 7 and 6 respectively. Zero for λ means LDA model without integrating link information.

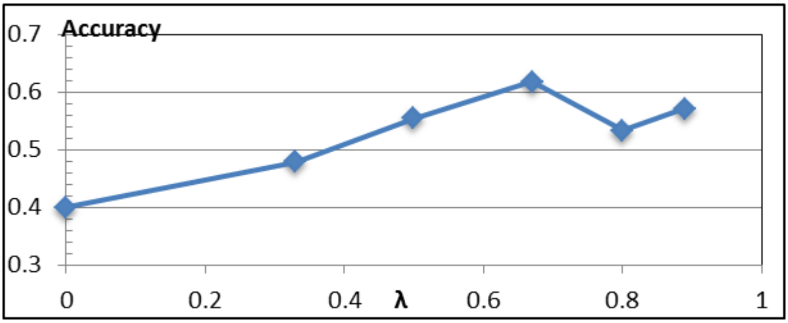


Fig. 5. Accuracy for Cora dataset. λ is set to 0, 0.33, 0.5, 0.67, 0.8,0.89 respectively. These values correspond to different ratios for existed literature and content of the document, which are 2:1, 1:1, 1:2, 1:4, 1:8.

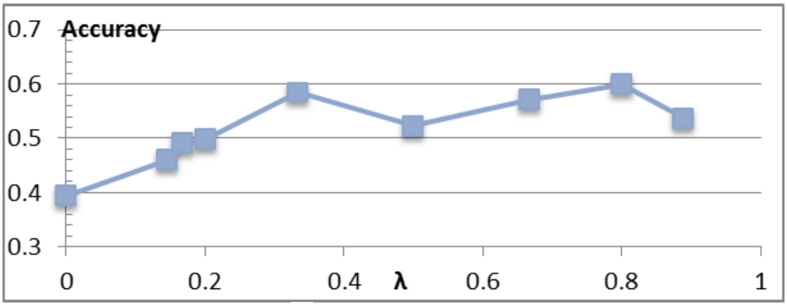


Fig. 6. Accuracy for Citeseer dataset. λ is set to 0, 0.145, 0.167, 0.2, 0.33, 0.5, 0.67, 0.8, 0.89 respectively. These values correspond to different ratios for existing literature and content of the document, which are 6:1, 5:1, 4:1, 2:1, 1:1, 1:2, 1:4, 1:8.

As depicted in Figure 5 and Figure 6, our model outperforms baseline greatly in all situations with different tuning parameters. Integrating link information into topic model reforms the document representation in latent semantic space.

Besides this, we can also observe the influence on model by the tuning parameter λ. Our experiments show that different datasets have different best tuning parameter values. We observe that there may exist two best tuning parameter values for a single dataset as shown in Figure 5 and Figure 6. It is interesting for it indicates the research's writing habits. The tuning parameter actually reveals the balance point between individuals and the population. The optimal tuning parameters correspond to the balance points for them. The first optimal value for λ is what we want to have. It indicates to what extent a researcher would gain knowledge from existed literature while doing research and writing papers. This value maintains researchers' individual characteristics and the population's features. For example, the best value for tuning parameter is 0.67 in Cora dataset. In other words, for every three words in the document of Cora, there is only one word that only talks about the authors' ideas without anything for the existed knowledge in the literature [Figure 7]. And for Citeseer, the value is 0.33. It is smaller than that in Cora dataset. Cora dataset consists of papers from machine learning area and it is reasonable for that papers of Cora dataset have more similar topics to each other and they have more coverage of each other. The second optimal value for λ corresponds to a different situation. In this situation, the authors are completely influenced by the existed literature. No innovations take place. To show this in the topic model, all of the documents are generated by the same topic distribution in the latent semantic space. That means all of the documents have the same document specific distributions over topics. Obviously, this is not we want to get for this would remove the diversity for the documents in the dataset.

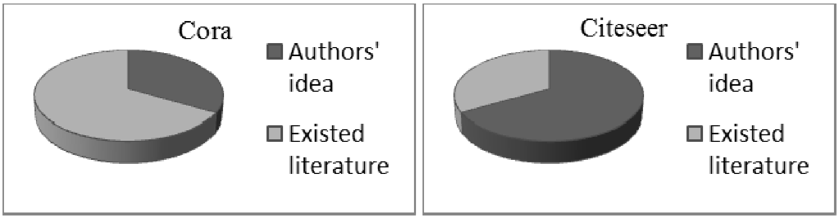


Fig. 7. Best tuning parameter for Cora and Citeseer datasets . The deep gray piece represents the authors' innovations. It indicates how different the author's idea from others works. The best tuning parameter maintains the diversity for the population and the characteristic for the individuals.

Although Cora and Citeseer contain several categories, these categories are also highly correlated to each other. For Cora, all of the documents belong to machine learning area. To see to what extent our model can distinguish them, we list the recall number in Table 2 and Table 3. Here, recall number is defined as the maximum number clustered in one category. Documents in these categories are more closely clustered compared with the procedure without utilizing the citations as prior information.

Table 2. Explicit recall number for Cora. C1 means represents category 1, etc. Row total means how many documents one category has. The star notation indicates the maximum recall number for a category.

	C1	C2	C3	C4	C5	C6	C7
Total	818	180	217	426	351	418	298
$\boldsymbol{\lambda = 0}$	291	72	130	**285***	93	170	151
$\boldsymbol{\lambda = 0.33}$	295	93	154	263	103	300	165
$\boldsymbol{\lambda = 0.5}$	401	94	144	253	221	280	**187***
$\boldsymbol{\lambda = 0.67}$	394	112	**157***	275	**228***	312	183
$\boldsymbol{\lambda = 0.8}$	397	119	126	279	208	329	179
$\boldsymbol{\lambda = 0.89}$	**416***	**123***	131	261	199	**337***	181

Table 3. Explicit recall number for Citeseer. C1 means represents category 1, etc. Row total means how many documents one category has. The star notation indicates the maximum recall number for a category.

	C1	C2	C3	C4	C5	C6
Total	596	668	701	249	508	590
$\boldsymbol{\lambda = 0}$	187	326	210	19	318	246
$\boldsymbol{\lambda = 0.145}$	350	329	211	45	303	288
$\boldsymbol{\lambda = 0.167}$	401	403	409	66	112	235
$\boldsymbol{\lambda = 0.2}$	249	434	397	69	275	226
$\boldsymbol{\lambda = 0.33}$	430	345	406	58	318	**379***
$\boldsymbol{\lambda = 0.5}$	349	**443***	414	20	279	226
$\boldsymbol{\lambda = 0.667}$	374	399	**443***	**71***	349	256
$\boldsymbol{\lambda = 0.8}$	435	393	370	61	**366***	362
$\boldsymbol{\lambda = 0.889}$	**450***	371	267	63	263	366

We can observe that recall number are significantly improved after integrating link information into the model. The bold columns are the significant ones. However, we also find that there are limiting values for recall numbers. This is restricted by the dataset itself. High limiting recall percentage means that documents in the category are highly related to each other and closely located together in the semantic space. Low limiting recall percentage means that the documents in the category cover many topics and are not well classified. Besides, for each category of the two datasets, the best tuning parameters are different. This phenomenon reveals that each category has individual cluster aggregation characteristics.

4.2.2 Perplexity

In this part, we measure how well our model performs in terms of perplexity. Perplexity is an important measurement in information theory. It is a common way of evaluating language models. The lower the perplexity is, the better the model trains the dataset. The perplexity formula is as follows:

$$\text{perplexity} = \sum_{d\in D} 2^{-\sum_{i=1}^{N}\frac{1}{N}\log_2 p(w|d)} \tag{5}$$

where p(w|d) represents the probability of the document generating a specific word:

$$\text{p(w|d)} = \sum_{z\in Z} p(z|d)p(w|z) \tag{6}$$

Formula 5 list calculate the total perplexity for the corpus. To get average perplexity for each document, we have to divide them by dataset size which is 2708 for Cora and 3312 for Citeseer respectively.

Perplexities of the two datasets for all the models in section 4.2.1 are listed in Table 4.

Table 4. Perplexities for Cora and Citeseer datasets under different conditions. The bold values in Table 4 means that it achieve a lower perplexity than baseline. For Cora, we can see that, in all 5 situations, we have lower perplexities. And for Citeseer, there are also 5 situations when we get a lower perplexity. For both datasets, when take the first optimal value for λ, which is 0.667 for Cora and 0.33 for Citeseer, we also get a lower perplexity than baseline.

λ	*Cora*	*Citeseer*
0	223070.4	190888.6
0.145		190972.7
0.167		***189718.6***
0.2		***188660.7***
0.33	***218606***	***190091.4***
0.5	***220083.9***	***189109***
0.667	***219184.8***	***187625.4***
0.8	***218958.2***	191009.1
0.889	***218178.2***	190909.4

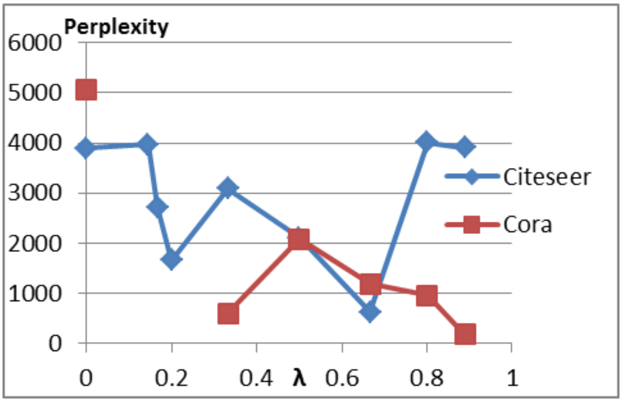


Fig. 8. Perplexities for Cora and CiteSeer datasets. To depict more clearly, we have already minus 218000 for Cora and 187000 respectively from the perplexities.

4.2.3 Citation Recommendation

We also manually evaluate the document recommendation performance of our model. The crucial part of recommending documents is to measure the similarity between documents. For example, we take a paper titled "Modeling Risk from a Disease in Time and Space" from the Cora dataset. This paper is mainly about Bayes network and Markov chain Monte Carlo (MCMC) methods cover most part of it.

Table 5. Example of recommending citations.For the paper titled "Modeling Risk from a Disease in Time and Space" in Cora, several citations recommend by our model are listed.

Modeling Risk from a Disease in Time and Space	
Citations	Recommended documents
1. Bayesian Dynamic Factor Models and Portfolio Allocation	1. On MCMC sampling in hierarchical longitudinal models
2. Bayesian Analysis of Agricultural Field Experiments	2. Exact bound for the convergence of metropolis chains
3. Hierarchical Spatio-Temporal Mapping of Disease Rates	3. A simulation approach to convergence rates for Markov chain Monte Carlo algorithms

We can represent this particular paper, its citations and recommended documents in the following composition chart shown in Figure 9. Usually papers consist of composition 1 style and composition 2 style are strongly correlated due to their highly similar compositions. Composition 3 style is less likely regarded as strongly related to composition 1 by this criterion, although its main part concerns the same topic, such as MCMC in this example. As discussed above, our model slightly removes the common parts, which is the population features, from its distribution. As a result, composition 3 would be more related to the refined distribution of composition 1. The recommended documents listed in the right part of Table 5 are strongly related to MCMC and recommending them as similar documents is reasonable.

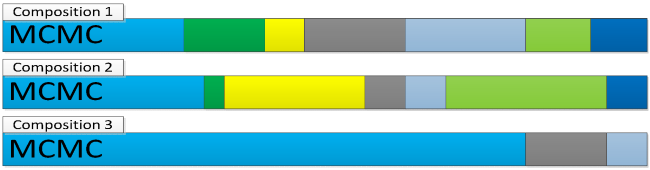


Fig. 9. Different types of compositions for documents. Different colors mean different topics. And the length of box indicate the topic weight in the distribution.

5 Related Work

Research on how to utilize links can be categorized into two groups. The first one is that links are utilized as degrees in graph theory. The other one is that links are treated as attributes of documents just like word appearance in the document. Trevor

Strohman [3] did a survey on the impact of all attributes in the documents. Among these attributes, such as publication year, text similarity, co-citation coupling, Katz distance and citation count, text similarity and Katz distance play the most important part. They are the two key attributes for a document. Much work has been done to integrate these two parts together to help the research.

On the basis of LDA, which analyze the document content in a low latent topic space, many derivations of LDA are proposed to tackle this problem. Cohn and Hoffman [4] proposed an extension to the pLSA [6] model, which called PHITS. Citations are modeled with words simultaneously and they are treated equally. Both of them share the same latent topic distribution. The intuition is that topic related documents have more intersection not only of words but also of citations. So citations or hyperlinks will be helpful in modeling the documents more precisely, which will eventually improve the performance using these distributions over latent topics. Link-LDA model [5] is very similar to PHITS. Erosheva et al developed PHITS by replacing pLSA with LDA. Reference sampling process is exactly the same to word sampling process. Both PHITS and Link-LDA model treat citations as word appearance. The generation process is completely guaranteed by the document specific topic distribution. They are all treated as observations while maximizing the likelihood function. However, documents dependences which revealed by citations are ignored in these models. Therefore, some other models were proposed to address this problem. Nallapati proposed Pairwise link-LDA and Link-PLSI-LDA [7] to tackle the document dependency problem. In this model, citations are guaranteed by document pair's topic distribution. For each pair of documents, it is treated as presence or absence of a citation which depends on a Bernoulli random variable. To explicitly consider the document relations represented by citations, Guo et al [2] proposed CT model which assumes a probabilistic generative process for corpora. Word sampling process in this model is completely controlled by the topic distribution of its citations. So the original content of the document itself is ignored. This perspective of treating citations can greatly reveal the document relations among them. Tang and Zhang [8] proposed a two layer Restricted Boltzmann Machines to model the links and word simultaneously. Links and words are linked together by a layer in the undirected graphical model.

Besides these topic models, many non-topical procedures are proposed. Qi He [9, 10] proposed another representation for document by utilizing the links information. They represent documents by its citation information. Citation information are actually manually generated contents by different researchers to describe a certain document. Therefore, citation information according to context are ideal for representing the documents by less words. Aya [11] proposed a machine learning algorithm to understand the motivation for the citations. Huang [12] investigated the effect citation contexts have when applied to clustering citations into topics and Ritachie [13] extensively investigated the impact of various citation context extraction methods.

Our procedure is different from previous procedures on how to treat links. We treat link as prior information in another way instead of word appearances. By doing this, we can maintain the dependencies while we model the documents. Both topical similarities between and dependencies documents are reflected in our proposed Plink-LDA model, which in turn promotes the performance. We compare our results on dataset Cora with Zhen Guo [2] procedure. Both of our procedures outperform the baseline, LDA procedure. The accuracy is around 40% for LDA, 47% for their procedure and 62% for our Plink-LDA model which is shown in Figure 5.

6 Conclusion

In this paper, we explore the feasibility of utilizing citation information in another way. We propose a model which models citations and words simultaneously. In our model, citations are no longer regarded as observations but prior information. We evaluate this model and the results show that it is feasible. Besides, the proposed model can find the researchers' writing habits in the dataset.

In the future, we plan to explore the problem how to determine the tuning parameters automatically for different datasets, such as using EM algorithms.

Acknowledgements. The work is supported by the Natural Science Foundation of China No. 61035004, No. 60973102 and THU-NUS NExT Co-Lab.

References

1. Blei, D., Ng, A., Jordan, M.: Latent Dirichlet Allocation. Journal of Machine Learning Research 3, 993–1022 (2003)
2. Guo, Z., Zhu, S., Chi, Y., Zhang, Z., Gong, Y.: A latent topic model for linked documents. In: Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, Boston, MA, USA, July 19-23, pp. 720–721. ACM (2009), 978-1-60558-483-6
3. Strohman, T., Bruce Croft, W., Jensen, D.: Recommending citations for academic papers. In: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2007, Amsterdam, The Netherlands, July 23-27, pp. 705–706. ACM (2007), 978-1-59593-597-7
4. Cohn, D.A., Hofmann, T.: The Missing Link - A Probabilistic Model of Document Content and Hypertext Connectivity. In: Advances in Neural Information Processing Systems, Denver, CO, USA, vol. 13, pp. 430–436. MIT Press (2000), Papers from Neural Information Processing Systems (NIPS) (2000)
5. Erosheva, E., Fienberg, S., Lafferty, J.: Mixed-membership models of scientific publications. Proceedings of the National Academy of Sciences, Sci. USA 101, 5220–5227 (2004)
6. Hofmann, T.: Probabilistic latent semantic indexing. In: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1999, Berkeley, CA, USA, August 15-19, pp. 50–57. ACM (1999)
7. Nallapati, R., Ahmed, A., Xing, E.P., Cohen, W.W.: Joint Latent Topic Models for Text and Citations. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, pp. 542–550. ACM (2008), 978-1-60558-193-4
8. Tang, J., Zhang, J.: A Discriminative Approach to Topic-based Citation Recommendation. In: Theeramunkong, T., Kijsirikul, B., Cercone, N., Ho, T.-B. (eds.) PAKDD 2009. LNCS, vol. 5476, pp. 572–579. Springer, Heidelberg (2009), ISSN: 978-3-642-01306-5

9. He, Q., Pei, J., Kifer, D., Mitra, P., Lee Giles, C.: Context-aware Citation Recommendation. In: Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, pp. 421–430. ACM (2010), 978-1-60558-799-8
10. He, Q., Kifer, D., Pei, J., Mitra, P., Lee Giles, C.: Citation Recommendation without Author Supervision. In: Proceedings of the Forth International Conference on Web Search and Web Data Mining, WSDM 2011, Hong Kong, China, February 9-12, pp. 755–764. ACM (2011), 978-1-4503-0493-1
11. Aya, S., Lagoze, C., Joachims, T.: Citation Classification and its Applications. In: Proceedings of the 2005 International Conference on Knowledge Management, ICKM 2005, North Carolina, USA, October 27-28, pp. 287–298 (2005)
12. Huang, S., Xue, G.-R., Zhang, B., Chen, Z., Yu, Y., Ma, W.-Y.: TSSP: A Reinforcement Algorithm to Find Related Papers. In: 2004 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2004), Beijing, China, September 20-24, pp. 117–123. IEEE Computer Society (2004), 0-7695-2100-2
13. Ritchie, A.: Citation context analysis for information retrieval. PhD thesis, University of Cambridge (2008)
14. Liu, Y., Niculescu-Mizil, A., Gryc, W.: Topic-Link LDA: Joint models of topic and author community. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, p. 84. ACM (2009), 978-1-60558-516-1

AnyOut: Anytime Outlier Detection on Streaming Data

Ira Assent[1], Philipp Kranen[2], Corinna Baldauf[2], and Thomas Seidl[2]

[1] Dept. of Computer Science, Aarhus University, Denmark
[2] Data Management and Data Exploration Group,
RWTH Aachen University, Germany

Abstract. With the increase of sensor and monitoring applications, data mining on streaming data is receiving increasing research attention. As data is continuously generated, mining algorithms need to be able to analyze the data in a one-pass fashion. In many applications the rate at which the data objects arrive varies greatly. This has led to anytime mining algorithms for classification or clustering. They successfully mine data until the a priori unknown point of interruption by the next data in the stream.

In this work we investigate anytime outlier detection. Anytime outlier detection denotes the problem of determining within any period of time whether an object in a data stream is anomalous. The more time is available, the more reliable the decision should be. We introduce AnyOut, an algorithm capable of solving anytime outlier detection, and investigate different approaches to build up the underlying data structure. We propose a confidence measure for AnyOut that allows to improve the performance on constant data streams. We evaluate our method in thorough experiments and demonstrate its performance in comparison with established algorithms for outlier detection.

1 Introduction

Wide availability of sensors, surveillance and measuring technology has lead to a remarkable increase in streaming data. Streaming data is typically continuously collected, and thereby needs to be analyzed in a one-pass manner as the data arrives. This is in contrast to traditional databases, where data may be accessed randomly and more than once.

Existing stream mining algorithms, usually assume constant streams in the sense that the time between the arrival of any two objects is fixed. This assumption does not hold for applications that collect data as and when required. For example, sensor networks minimize energy consumption and communication overhead by sending information to a server only in the case of events or changes. As a consequence, the time for analyzing the data stream may vary greatly. When a burst occurs, little time is available. When stream speed is slow, the additional time should be used to improve accuracy. This type of behavior characterizes the new family of anytime mining algorithms, which received considerable research attention for clustering and classification [18,29,14,38,11].

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 228–242, 2012.
© Springer-Verlag Berlin Heidelberg 2012

Outlier detection has been defined as finding "an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" [22]. In this work we investigate the anytime outlier detection problem. We introduce our AnyOut technique, which uses a cluster-based approach to represent data in a hierarchical fashion. As long as time is available, the hierarchy is traversed to determine an outlier score based on the deviation between object and cluster. Until interrupted, more fine grained resolution of the clustering structure is analyzed. We discussed a sketch of the basic idea in a short workshop paper [5]. Here we investigate two ways to build up the tree structure and propose a confidence measure for our algorithm to harness its strength also for constant data streams. Besides the novel confidence measure and construction method a major contribution is a thorough evaluation against established baseline algorithms. We show how AnyOut successfully detects outliers in various streams settings and compare its performance to LOF, ABOD and OPTICSOF.

2 Related Work

Outlier detection is sometimes studied as a supervised problem, i.e., similar to unbalanced classification [45,16]. Supervised methods require labeled outliers as training data. This is often not the case in practice.

Unsupervised approaches do not assume labeled training data, but identify outliers based on their deviation from the remainder of the data. In statistical outlier detection, it is assumed that the data follows a certain distribution, and objects that do not well fit this assumption are outliers [6,35,36]. Distance-based outlier detection finds objects that show a high distance to most other objects [27]. Clustering-based outlier detection uses clusters to identify the inherent structure of valid data, and finds objects that are not clustered well [15,23]. While many methods separate outliers and inliers (valid data), finding a clear boundary may prove difficult. The Local Outlier Factor (LOF) method relaxes this requirement in favor of a scoring function that reflects the degree of deviation [9]. The result is then not a set of outliers, but a ranking by degree of outlierness.

Also for time series data, outlier detection has been proposed [37]. Time series are sequences of values in temporal order. Please note that while this bears some resemblance with streaming data, the important difference is that time series are assumed to be available entirely at the time of outlier detection. Moreover, the goal is not to identify outlying objects as in data streams, but outlying patterns of several temporally neighboring values. Thus, time series outlier detection is different from outlier detection in data streams.

Recently, approaches for outlier detection on data streams have been proposed [2,4,40,41,17,43,25,42,10]. However, all of these approaches assume streams of fixed arrival rates and do not meet the requirements of anytime outlier detection: interruptible and improvement of accuracy with more time.

Anytime algorithms have been studied in artificial intelligence [28]. In anytime learning, the training phase for supervised learning is restricted [18,14].

Other approaches monitor the performance of anytime algorithms [21]. Anytime clustering is addressed by [29], anytime classification by [11,38]. We presented a preliminary version of our anytime outlier concept at a KDD workshop [5].

3 Detecting Outliers in Streaming Data

In outlier detection, the goal is to identify objects which deviate from the remainder of the data. Here, we abstract from the concrete notion of what constitutes an outlier in order to formalize the anytime outlier detection problem. Different outlier detection paradigms may be followed in order to address this problem. In Subsections 3.1 and 3.2, we propose a cluster-based solution.

As briefly sketched in Related Work as well, it may prove difficult to determine clear decision boundaries between outliers and inliers. Typically, objects deviate from the majority of the data objects to a varying degree. Consequently, many outlier detection techniques aim to capture this degree of deviation in a corresponding outlier score (e.g. LOF [9]). We assume that outlier detection is concerned with the computation of an outlier score that expresses the degree of outlierness. Please note that this is without loss of generality, as algorithms that separate outliers and inliers can be considered binary special cases.

Common for all approaches on streaming data is that they process the data as it arrives. Existing work on streaming outlier detection, however, has assumed that the stream of incoming data objects is of constant speed, i.e. the time that passes between any two objects arriving in the stream is constant for the entire stream. In many applications, e.g. in sensor networks that are optimized for low power consumption and communication overhead, data is generated depending on changes in the outside environment. These streams are not of constant, but of varying inter-arrival rates. Consequently, the amount of time that is available for outlier detection varies as well.

As illustrated in Figure 1, as more objects arrive within shorter periods of time, decision on outlier detection needs to be taken in less time accordingly. Please note that since the duration of a burst or the number of objects that arrive within a certain time window is generally not bounded, simply buffering objects until the stream slows down is not an option. If the buffer size is exceeded, objects are lost and detection accuracy drops unexpectedly and unpredictably. On the other hand, if the stream speed is slow, an ideal outlier detection algorithm should be capable of making use of this time in order to improve accuracy.

A naive solution to the requirement of anytime behavior might be to resort to a number of outlier detection methods ordered by their reliability. These could be accessed one by one until this collection of methods is interrupted. Clearly,

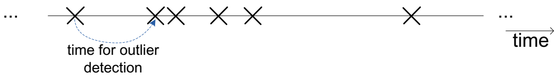


Fig. 1. Streaming data with varying inter-arrival rates

this method does not make best use of the time available as each method would have to start over from scratch.

An anytime outlier detection approach should thus be capable of

- fully using available time (more time leads to more accurate results)
- returning a result anytime (meaningful interruption as objects arrive)
- continuing computation incrementally (not simply starting over).

Definition 1. ***Anytime outlier detection.*** *Given a data stream of data objects o_i arriving at a priori unknown inter-arrival rate, the anytime outlier detection problem is to compute an outlier score $s(o_i)$ in the time t_i between the arrival of o_i and its successor o_{i+1}. The larger t_i, the more accurate the outlier score $s(o_i)$ should be.*

Please note that evaluation of the accuracy of outlier scores $s(o_i)$ is not straightforward. If synthetic or manually labeled data is available for empirical studies, this constitutes a ground truth for accuracy assessment. In practice, however, such ground truth is typically not available, and domain experts need to judge the quality of the outlier scores that are assigned to objects in the stream. This issue is not specific to streaming environments, but affects outlier detection research in general.

3.1 AnyOut : Overview over Our Method

In our AnyOut method, we propose following a cluster-based approached for outlier detection. As mentioned in the related work section, clustering methods have been successfully used to identify prevailing patterns of the data that serve as an input to the actual outlier detection.

In order to meet the requirements of fast initial response and improvement over time, we suggest using a hierarchy of clusters in a tree structure that is traversed until the algorithm is interrupted by the next arriving data object in the stream. Clusters at upper levels of the tree hierarchy subsume the more fine grained information at lower levels of the tree. The hierarchy of the tree thereby provides a natural organization of the clusters that can be incrementally accessed in order to refine the outlier score of the object in question. Initially, the object is compared only to the root node that describes the data distribution using few clusters. This comparison can be performed efficiently, thereby meeting the first requirement. Since more detailed information on the data distribution is available at lower levels of the tree, the reliability of outlier scores is typically improved. This aspect is empirically studied in the experiments, in Section 4.

3.2 Outlier Detection Using a Cluster Hierarchy

We introduced the ClusTree for clustering [29] as an extension to the R-tree family [19,38]. The core concept is the use of nodes to compactly represent clusters using cluster features. A cluster feature $CF = (n, LS, SS)$ is a tuple of the number of objects in the cluster n, of the linear sum of these objects LS, and of their squared sum SS. Please note that both LS and SS are vectors of the same dimensionality as the data. The information in the tuple of a cluster feature CF suffices for the computation of cluster properties such as the mean and the variance of the objects within the cluster. Cluster features have been successfully used to summarize clusters in several existing works [3,44,29]. The ClusTree is created and updated like any multidimensional index structure. As an extension to these multidimensional indexes, the ClusTree additional provides buffer entries that are used for anytime insertion of clusters. Since anytime clustering is not the focus of this work, details on these buffers are omitted here. Interested readers are referred to [29].

Figure 2 presents an overview over the ClusTree structure. A node in this illustration consists of two entries that each contain a cluster feature (depicted here as a curve that represents the data distribution within the cluster), a pointer to child nodes and a buffer that might contain a cluster feature as well. Pointers to child nodes are used for navigation during anytime processing to go to finer representations of a particular cluster.

Given a data set of objects, the question is how to create the tree structure. While one could simply follow the bottom-up insertion method, a better structure can be achieved using top-down tree creation methods, also termed bulk loading in the indexing community. The general idea is to initialize the indexing structure such that data within a node is closely related. In this work, we follow a bulk loading method presented in [32] using the EM (expectation maximization) algorithm [12].

For anytime outlier detection, we propose using the structure of the ClusTree in order to perform cluster-based outlier score computation. The idea is to compare the object to clusters at the levels of the tree in a top-down fashion as long as the time allowance dictated by the stream permits.

Assessing the degree of outlierness in our AnyOut approach is based on the degree of accordance of the object with the closest cluster feature at the current level of resolution. Whenever a new object o_{i+1} arrives in the stream, this interrupts the descent down the tree with the current object o_i. We then compare the object o_i to the cluster feature to assess the outlier degree.

In order to define the actual outlier score, we reinvestigate the information stored in the cluster features. As mentioned above, cluster features provide sufficient information to compute statistical properties of the objects within the

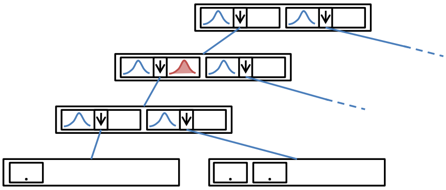


Fig. 2. ClusTree: Node entries represent clusters(descriptor, child pointer, buffer); child nodes more fine grained clusters of the parent data

cluster. Thus, objects in a cluster can be interpreted as observations of the data distribution in this part of feature space. Based on this intuition, we propose a *mean outlier score* that computes the degree of outlierness of an object o_i as the extent of deviation of o_i from the mean of the closest cluster feature. Since AnyOut operates directly on the ClusTree, the closest cluster feature is simply the one that is reached through tree traversal until the point of interruption by the next data object in the stream.

Definition 2. ***Mean outlier score.*** *For any data object o_i, the mean outlier score $s_m(o_i)$ is defined as $s_m(o_i) := dist(o_i, \mu(e_s))$, where $\mu(e_s)$ is the mean of entry e_s in the ClusTree that o_i is inserted into when the next object o_{i+1} of the data stream arrives.*

The mean outlier score thus assesses the deviation of the current object from the mean of the data distribution in the cluster feature of the current tree entry.

We propose a second way of assessing the outlierness that further extends the notion that cluster features represent the data distribution. By interpreting the cluster features as parameters of a Gaussian distribution of the data objects in this subtree, we arrive at a second outlierness score based on the density of the object. Recall that the Gaussian probability density of an object o_i for mean μ_{e_s} and covariance matrix Σ_{e_s} of an entry e_s is computed as

$$g(o_i, e_s) = \frac{1}{(2\pi)^{d/2} \cdot det(\Sigma_{e_s})^{1/2}} e^{\left(-\frac{1}{2}(o_i - \mu_{e_s})^T \Sigma_{e_s}^{-1} (o_i - \mu_{e_s})\right)} \tag{1}$$

where $det(\Sigma_{e_s})$ is the determinant and $\Sigma_{e_s}^{-1}$ the inverse of Σ_{e_s}. Please note that we can estimate the probability density of o_i on e_s based exclusively on e_s's cluster feature. While the full covariance is not available using a cluster feature (storing covariances is of quadratic complexity instead of linear complexity), using a matrix of variances has been empirically shown to achieve high accuracy results e.g. in classification [38]. Thus, a cluster feature is sufficient to compute an outlier score that reflects the density of the object with respect to the data distribution.

Definition 3. ***Density outlier score.*** *For any data object o_i, the density outlier score $s_d(o_i)$ is defined as $s_d(o_i) := 1 - g(o_i, e_s)$ (cf. Equation 1), where e_s is the entry in the ClusTree that o_i is inserted into when the next object o_{i+1} of the data stream arrives.*

Both the mean outlier score and the density outlier score reflect the degree of outlierness of the object at the point of interruption. In both cases, the data distribution of the closest cluster is the basis for the score computation. They differ in the way the cluster feature is interpreted: the mean outlier score only takes the center of mass of the data into account, whereas the density outlier score takes the overall data distribution into account, assuming a Gaussian distribution.

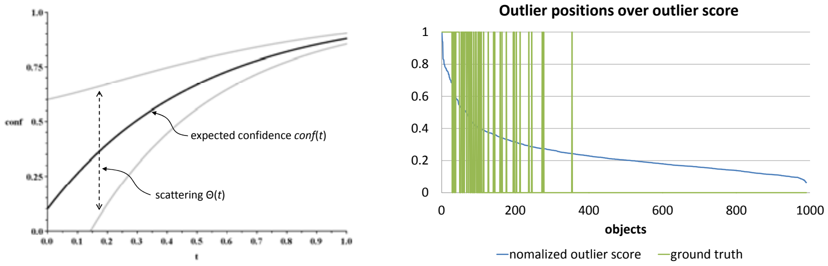


Fig. 3. Left: Expected confidence and scattering. Right: Actual outlier positions.

3.3 AnyOut Confidence Measure for Constant Streams

The benefit of anytime algorithms on constant data streams has first been shown in [33]. The scenario is as follows: the items arrive in a constant time interval t_a, the features are taken at time t_f and a decision must be available at time t_d. Instead of computing each object consecutively for a constant time t_a, the approach takes a window of several objects and processes these in parallel. Thereby it can freely distribute the available time among the objects, i.e. it can spend less time on *simple* cases and more time on more *difficult* cases.

The core point is the notion of simple and difficult, which is handled using a confidence measure. All objects are initialized and for each object a confidence value for the current result is computed that indicates how certain the algorithm is with the current output for that object. Just after the initialization the expected confidence value is rather low. If more time is used for computation, the expected confidence value increases. However, the actual confidence values will be scattered around the expected value, i.e. some items have a higher confidence and other lower than expected. The amount of scattering decreases with increasing computation time (cf. Figure 3 left). The scattered confidences are used to distribute the computation time, i.e. the item with the least confidence value is allowed to improve its result by one step (e.g. descending one more level in the AnyOut algorithm). This process is repeated until the time is used up, i.e. until the decision for the first item has to made. At that time the next window of items is taken to be processed in parallel. A second approach was proposed in [33], which uses a FiFo queue that contains all objects between t_f and t_d. We will use both approaches to evaluate AnyOut , for more details refer to [33].

To test the performance of the AnyOut algorithm on constant data streams we hence need a confidence measure for the current outlier score of an object. To this end we make use of the positions of the outliers in the ranking returned by the AnyOut algorithm. Figure 3 shows the results for the vowel data set (middle level of the tree, leaving out one class as outliers, cf. Section 4) using the mean outlier score. Optimally all outliers (green bars) would be on the far left. Obviously the objects with the highest outlier score are false alarms. Similar distributions are observable for other data sets. Therefore we use

$$conf(o) = e^{-s(o)}$$

as a simple and straight forward confidence measure, where $s(o)$ is the current outlier score of object o. Intuitively, we check for the putative outliers, i.e. the highest ranked objects, whether they are really outliers by giving them more computation time.

4 Experiments

In the following we first analyze the performance of our proposed AnyOut algorithm with respect to the individual levels of the tree structure as well as on different stream settings. In Section 4.3 we then compare our performance against established baseline methods, namely LOF, ABOD and OPTICSOF. We use the implementations of these algorithms provided in the ELKI framework [1], which is publicly available from the ELKI homepage[1]. Finally we discuss results for data streams containing drift and novelty in Section 4.4. All experiments use real world data sets of different characteristics from the UCI machine learning repository [24].

We evaluate the algorithms after all objects are processed, thereby we have a complete ranking of all objects with respect to their assigned outlier score in descending order. We use the ranking and the ground truth to compute the ROC curve for the results, which plots the true positive rate (TPR) over the false positive rate for each prefix of the ranking. From the ROC plots we derive the AUC value (area under the ROC curve [8]) as a first measure. We compute two further standard measures for ranking quality, namely the Spearman ranking coefficient (SRC) [39] and Kendall's Tau [26].

4.1 Level Analysis

For all experiments in Sections 4.1 and 4.2 we perform four fold cross validation. More precisely, we build up the tree (incrementally or with the described bulk loading technique) using the training set and analyze its performance. Continuous learning is investigated in Sections 4.3 and 4.4. To generate outliers in the following we leave out one class in the training set, i.e. the objects from the left out class which are contained in the test set are the outliers $\mathcal{O}$. In the plots we report the averaged performance values over all classes and folds.

To analyze the individual levels of the resulting tree structures in our AnyOut algorithm, we created one ranking for each level of the tree and computed the measures introduced above. Figure 4 shows the ROC plots for the vowel, pendigits and letter data sets. Each plot contains a ROC curve for the root level, the leaf level and a level in the middle. The tree structures in this experiment were build using incremental insertion and the employed outlier score was the mean outlier score. What is clearly visible from the results on all data sets, is that the basic principle of the AnyOut algorithm is effective, i.e. the quality of the results improves if more time is available (deeper levels are reachable).

[1] ELKI project homepage: `http://elki.dbs.ifi.lmu.de/`

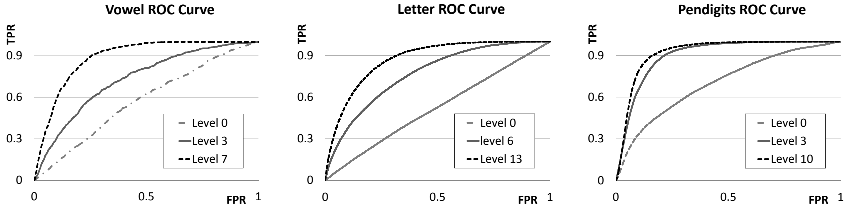


Fig. 4. ROC curves for the vowel, letter and pendigits data sets

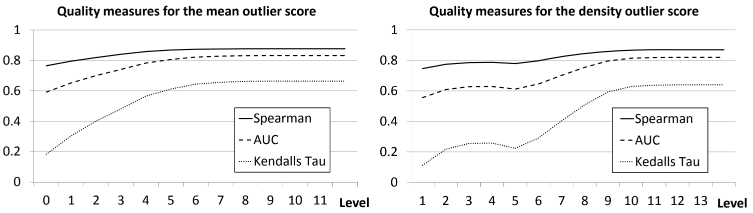


Fig. 5. Quality Measures for mean and density outlier scores on vowel

Similar observations can be made using the other quality measures. Figure 5 shows the results for Spearman's ranking coefficient, AUC and Kendalls Tau. The two plots compare the mean outlier score and the density outlier score on the vowel data set using bulk loading to construct the tree. (We compare incremental insertion and bulk loading in the next section.) The density outlier score yields slightly worse rankings than the mean outlier score. Moreover, the mean outlier score yields a constant quality improvement, whereas the density outlier score shows a slight reduction in ranking quality on intermediate levels. The mean outlier score showed better results throughout the data sets and we therefore employ it in the following.

4.2 AnyOut Performance on Various Stream Settings

To evaluate the anytime performance of AnyOut under variable stream scenarios, we recapitulate a stochastic model that is widely used to model random arrivals [13]. A Poisson process describes streams where the inter-arrival times are independently exponentially distributed. Poisson processes are parameterized by an arrival rate parameter λ:

Definition 4. ***Poisson stream.*** *A probability density function for the inter-arrival time of a Poisson process is exponentially distributed with parameter λ by*

$$p(t) = \lambda \cdot e^{-\lambda t}$$

The expected inter-arrival time of an exponentially distributed random variable with parameter λ is $E[t] = \frac{1}{\lambda}$.

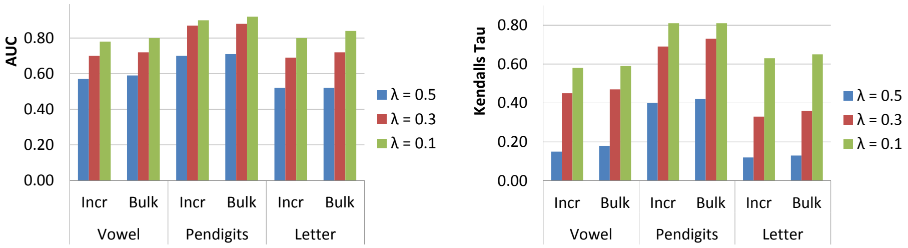


Fig. 6. Results on anytime streams

We use a Poisson process to model random stream arrivals for each fold of the cross validation. We randomly generate exponentially distributed inter-arrival times for different values of λ. If a new object arrives (the time between two objects has passed) we interrupt the AnyOut algorithm and store the outlier score for the object. We repeat this experiment using different expected inter-arrival times $\frac{1}{\lambda}$, where a unit corresponds to a level of the tree structure. We assume that any object arrives at the earliest after the initialization phase of the previous object, i.e. after evaluating the root of the tree.

Figure 6 shows the results for Poisson streams using the vowel, pendigits and letter data sets. In these experiments we compare the performance of the incremental insertion to the described bulk loading using the EM algorithm. In each group of bars the values for three different λ values are shown, while a smaller value corresponds to a slower stream according to Definition 4, i.e. more expected time per object.

Starting with the vowel data set in Figure 6 we see for the AUC measure and for Kendall's Tau, that both the incremental insertion and the bulk loading yield better qualities on slower streams. This is in line with the results from the level analysis in Section 4.1. Comparing the two tree construction methods we observe that on the one hand the bulk loading yields better results for any speed on both measures, but on the other hand the advantage is smaller than we expected.

The results on pendigits and letter (Figure 6) confirm both of the above findings. With a higher expected time per object (smaller λ value) the AnyOut algorithm shows its effectiveness and produces better rankings. The difference between incremental insertion and bulk loading performance are rather small but constant.

Summarizing the evaluation on varying data streams we can conclude that the proposed AnyOut method is effective for anytime outlier detection.

For constant streams, Figure 7 shows the corresponding results for the vowel data set using the window approach and FiFo approach introduced in Section 3.3. We evaluated different window and FiFo sizes from 2 to 12 (denoted as WS and FS in Figure 7). As above we performed four fold cross validation and computed the quality measures based on the final ranking of all objects. The results from Figure 7 (left) show that the performance of the AnyOut algorithm improves with larger

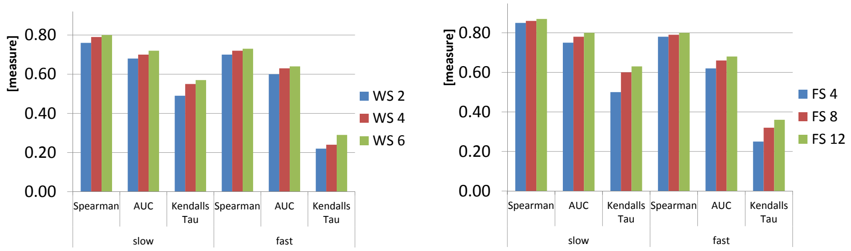


Fig. 7. AnyOut on constant data streams

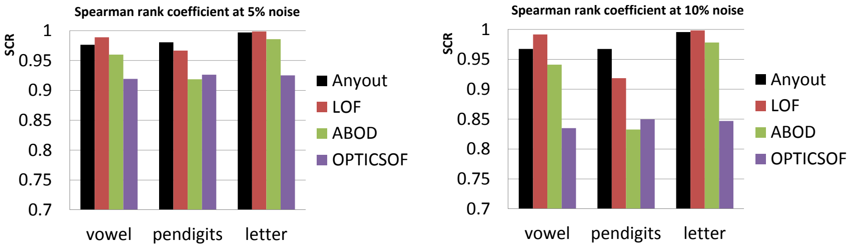


Fig. 8. Comparing AnyOut against baseline methods for 5% and 10% noise

window sizes for slow and fast streams. The results using the FiFo approach are even slightly better (cf. right part of Figure 7), which is attributable to the greater flexibility according to the comparably larger set of objects. This is in line with the results from [33] for anytime classification on constant streams and shows the effectiveness of the proposed confidence measure for the AnyOut algorithm.

4.3 Comparison to Baseline Methods

We assess the performance of AnyOut against established outlier detection methods LOF, ABOD and OPTICSOF from the ELKI framework on the same real world data sets. To generate outliers we added a certain percentage of noise uniformly distributed in the feature space. AnyOut processed the points one after the other incrementally inserting them into the tree structure and returning an outlier score. The competing approaches were given the entire data and were allowed random access. We tested k-values from 1 to 100 for LOF and OPTICSOF and report the best results per data set. This way we compare against the best possible values for AUC, SRC and Kendalls Tau from LOF, ABOD and OPTICSOF.

Figure 8 shows the resulting SRC values for all four approaches with 5 and 10 percent noise respectively. On vowel and letter the LOF algorithm achieves the highest SRC values for both noise settings. AnyOut and ABOD yield comparable yet slightly smaller SRC values and OPTICSOF follows at larger distance. On the pendigits data set AnyOut performs even slightly better than LOF, while the other two approaches perform at a lower level. The effects are slightly more pronounced at the higher noise level.

	Anyout				LOF				ABOD			
Noise	AUC	SRC	K. Tau	Time	AUC	SRC	K. Tau	Time	AUC	SRC	K. Tau	Time
5%	0.979	0.976	0.958	**24**	**0.996**	**0.989**	**0.992**	957	0.953	0.960	0.905	457
10%	0.979	0.967	0.958	**22**	**0.996**	**0.992**	**0.991**	1326	0.932	0.941	0.863	478

Fig. 9. Comparing AnyOut against baseline methods for different constraints on vowel

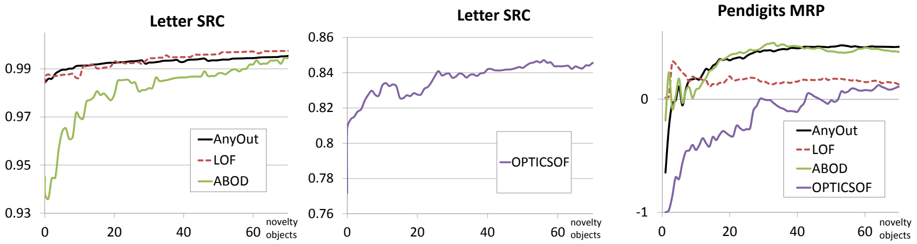


Fig. 10. Learning rate of the individual approaches in case of novelty

Figure 8 certifies AnyOut a competitive performance with respect to the Spearman rank coefficient. In Figure 9 we show the results on vowel for all three measures. Additionally we report the time in milliseconds that the algorithm used per object. As for the SRC in Figure 8, OPTICSOF performed worst and we spare the details for readability. Highlighted cells indicate best performance. LOF shows continuously best ranking values, closely followed by AnyOut and then ABOD at some distance. This is in line with the previous results. For the processing time we see that AnyOut is faster by at least one order of magnitude even for this small data set; on larger data sets the effect increases dramatically. This is however not surprising, since AnyOut is designed to be fast (logarithmic descent in the tree structure) while the competing approaches are not. Thus, we compare the ranking performance of our AnyOut against the best possible baseline: outlier approaches that are not capable of fast interruptible anytime processing.

4.4 Evolving Data Streams: Drift and Novelty

To test the performance of the algorithms for novelty and drift[2] in the data stream we used the same setup as in the previous section and left out one class in the first half of the stream. During the second half of the stream we evaluated the ranking after each object from the new class. The left part of Figure 10 shows the resulting SRC values for AnyOut, LOF and ABOD, the corresponding values for OPTICSOF are shown in the middle. ABOD shows the strongest reaction to the novel class. The SRC values are clearly lower for the first five novelty objects. After that they gradually increase until twenty objects from the novelty class have arrived. This effect is less pronounced for OPTICSOF, while the SRC values are generally lower (cf. Section 4.3). The reaction in terms of

[2] 'Novelty' refers to an entirely new concept (new class, new cluster), while 'Drift' refers to changes in the distribution of an existing concept.

SRC values is the smallest for LOF and AnyOut. Similar plots result for the other ranking measures. Since the reaction to an entirely new class is already very small, the reaction to gradually drifting concepts is negligible and not visible in the corresponding plots (not shown due to space limitations).

To further analyze the reasons for the minuscule reactions to novelty we investigate the positions of the novelty objects in the ranking of all objects. At any time we know the number n of outliers (noise objects). We normalize the ranking positions such that -1 indicates the highest rank (most probable outlier), 0 corresponds to the border object (n-th object) and 1 indicates the lowest rank (least probable outlier). As above, we evaluate the rankings after each novelty object. This time we compute the 'mean relative position' (MRP) of the novelty objects, i.e. the average of their ranking positions normalized as described above. The expected value for non-outlier objects is 0.5. Figure 10 shows the results for all four approaches on pendigits. OPTICSOF assigns the highest rank -1 to the first novelty object, after that the average rank value gradually increases as the algorithm learns the new concept. For LOF and ABOD the MRP hardly ever shows negative values. AnyOut still assigns a very high rank to the first novelty object (-0.65), but learns the new concept rapidly. After five to ten novelty objects the MRP value is above zero, i.e. the new class is no longer considered an outlier class.

In summary, AnyOut successfully learns novel concepts fast and handles drifting concepts, while being capable of anytime processing.

5 Conclusion and Future Work

In this paper we proposed the study of the anytime outlier detection problem for varying and constant data streams. We first analyzed and discussed the existing work on outlier detection and stream outlier detection before we formally defined the novel anytime outlier detection problem. We proposed a first algorithm called AnyOut long with two construction heuristics and a confidence measure for application with constant streams. We analyzed the structure and performance of AnyOut on various data streams using real world data. Finally we compared our method against established baseline algorithms finding comparable performance despite the largely reduced runtime. In future research more sophisticated confidence measures can be investigated as well as anytime outlier algorithms using other paradigms such as density based approaches.

The MOA framework [7] is an open source benchmarking software for data stream mining, similar to WEKA [20]. Recently we extended the MOA framework to support clustering and clustering evaluation on evolving data streams [30,34]. We now added stream outlier detection to the MOA framework [31] and integrated the AnyOut algorithm as well as a wrapper for the outlier methods provided by the ELKI framework [1].

Acknowledgments. This work has been supported by the UMIC Research Centre, RWTH Aachen University, Germany.

References

1. Achtert, E., Kriegel, H.-P., Reichert, L., Schubert, E., Wojdanowski, R., Zimek, A.: Visual Evaluation of Outlier Detection Models. In: Kitagawa, H., Ishikawa, Y., Li, Q., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 5982, pp. 396–399. Springer, Heidelberg (2010)
2. Aggarwal, C.C.: On abnormality detection in spuriously populated data streams. In: SDM (2005)
3. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: VLDB, pp. 81–92 (2003)
4. Angiulli, F., Fassetti, F.: Detecting distance-based outliers in streams of data. In: CIKM (2007)
5. Assent, I., Kranen, P., Baldauf, C., Seidl, T.: Detecting outliers on arbitrary data streams using anytime approaches. In: StreamKDD Workshop in Conjunction with 16th ACM SIGKDD (2010)
6. Barnett, V., Lewis, T.: Outliers in Statistical Data, 3rd edn. Wiley (1994)
7. Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T., Seidl, T.: Moa: Massive online analysis, a framework for stream classification and clustering. Journal of Machine Learning Research - Proceedings Track 11, 44–51 (2010)
8. Bradley, A.: The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognition 30(7), 1145–1159 (1997)
9. Breunig, M., Kriegel, H.-P., Ng, R., Sander, J.: LOF: Identifying density-based local outliers. In: ACM SIGMOD, pp. 93–104 (2000)
10. Cao, H., Zhou, Y., Shou, L., Chen, G.: Attribute Outlier Detection over Data Streams. In: Kitagawa, H., Ishikawa, Y., Li, Q., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 5982, pp. 216–230. Springer, Heidelberg (2010)
11. DeCoste, D.: Anytime query-tuned kernel machines via cholesky factorization. In: SDM (2003)
12. Dempster, A.P., Laird, N.M.L., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. J. Royal Stat. Soc. B 39(1), 1–38 (1977)
13. Duda, R., Hart, P., Stork, D.: Pattern Classification, 2nd edn. Wiley (2000)
14. Esmeir, S., Markovitch, S.: Interruptible anytime algorithms for iterative improvement of decision trees. In: UBDM Workshop at KDD (2005)
15. Ester, M., Kriegel, H.-P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases. In: KDD (1996)
16. Foss, A., Zaïane, O., Zilles, S.: Unsupervised Class Separation of Multivariate Data through Cumulative Variance-Based Ranking. In: ICDM (2009)
17. Franke, C., Gertz, M.: Detection and exploration of outlier regions in sensor data streams. In: ICDM Workshops, pp. 375–384 (2008)
18. Grefenstette, J., Ramsey, C.: An Approach to Anytime Learning. In: Workshop on Machine Learning, pp. 189–195 (1992)
19. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: ACM SIGMOD (1984)
20. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The weka data mining software: an update. SIGKDD Expl. Newsl. 11(1), 10–18 (2009)
21. Hansen, E.A., Zilberstein, S.: Monitoring anytime algorithms. SIGART Bulletin 7(2), 28–33 (1996)
22. Hawkins, D.: Identification of outliers. Chapman and Hall, New York (1980)
23. He, Z., Xu, X., Deng, S.: Discovering cluster-based local outliers. Pattern Recognition Letters (2003)

24. Hettich, S., Bay, S.: The UCI KDD archive (1999), http://kdd.ics.uci.edu
25. Hoang Vu, N., Gopalkrishnan, V., Namburi, P.: Online Outlier Detection Based on Relative Neighbourhood Dissimilarity. In: Bailey, J., Maier, D., Schewe, K.-D., Thalheim, B., Wang, X.S. (eds.) WISE 2008. LNCS, vol. 5175, pp. 50–61. Springer, Heidelberg (2008)
26. Kendall, M.: A new measure of rank correlation. Biometrika 30(1-2), 81 (1938)
27. Knorr, E., Ng, R., Tucakov, V.: Distance-based outliers: algorithms and applications. In: VLDBJ (2000)
28. Kotenko, I., Stankevitch, L.: The control of teams of autonomous objects in the time-constrained environments. In: ICTAI, pp. 158–163 (2002)
29. Kranen, P., Assent, I., Baldauf, C., Seidl, T.: Self-adaptive anytime stream clustering. In: ICDM (2009)
30. Kranen, P., Kremer, H., Jansen, T., Seidl, T., Bifet, A., Holmes, G., Pfahringer, B.: Clustering performance on evolving data streams: Assessing algorithms and evaluation measures within moa. In: ICDM (2010)
31. Kranen, P., Kremer, H., Jansen, T., Seidl, T., Bifet, A., Holmes, G., Pfahringer, B., Read, J.: Stream Data Mining using the MOA Framework. In: Lee, S.-G., et al. (eds.) DASFAA 2012, Part II. LNCS, vol. 7239, pp. 309–313. Springer, Heidelberg (2012)
32. P. Kranen and T. Seidl. Harnessing the strengths of anytime algorithms for constant data streams. *DMKD Journal (19)2, ECML PKDD Special Issue*, 2009.
33. Kranen, P., Seidl, T.: Harnessing the strengths of anytime algorithms for constant data streams. DMKD Journal 2(19) (2009) ECML PKDD Special Issue
34. Kremer, H., Kranen, P., Jansen, T., Seidl, T., Bifet, A., Holmes, G., Pfahringer, B.: An effective evaluation measure for clustering on evolving data streams. In: ACM SIGKDD, pp. 868–876 (2011)
35. Müller, E., Schiffer, M., Seidl, T.: Adaptive outlierness for subspace outlier ranking. In: CIKM, pp. 1629–1632. ACM (2010)
36. Müller, E., Schiffer, M., Seidl, T.: Statistical selection of relevant subspace projections for outlier ranking. In: ICDE, pp. 434–445. IEEE Computer Society (2011)
37. Muthukrishnan, S., Shah, R., Vitter, J.: Mining deviants in time series data streams. In: SSDBM (2004)
38. Seidl, T., Assent, I., Kranen, P., Krieger, R., Herrmann, J.: Indexing density models for incremental learning and anytime classification on data streams. In: EDBT (2009)
39. Spearman, C.: The Proof and Measurement of Association between Two Things. The American Journal of Psychology 15(1), 72–101 (1904)
40. Subramaniam, S., Palpanas, T., Papadopoulos, D., Kalogeraki, V., Gunopulos, D.: Online outlier detection in sensor data using non-parametric models. In: VLDB, pp. 187–198 (2006)
41. Yamanishi, K., Takeuchi, J., Williams, G., Milne, P.: On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. DMKD Journal 8(3), 275–300 (2004)
42. Yang, D., Rundensteiner, E.A., Ward, M.O.: Neighbor-based pattern detection for windows over streaming data. In: EDBT, pp. 529–540 (2009)
43. Zhang, J., Gao, Q., Wang, H.: Spot: A system for detecting projected outliers from high-dimensional data streams. In: ICDE, pp. 1628–1631 (2008)
44. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: an efficient data clustering method for very large databases. In: ACM SIGMOD (1996)
45. Zhu, C., Kitagawa, H., Faloutsos, C.: Example-Based Robust Outlier Detection in High Dimensional Datasets. In: ICDM (2005)

Ensemble Based Positive Unlabeled Learning for Time Series Classification

Minh Nhut Nguyen, Xiao-Li Li, and See-Kiong Ng

Institute for Infocomm Research, Singapore
{mnnguyen,xlli,skng}@i2r.a-star.edu.sg

Abstract. Many real-world applications in time series classification fall into the class of positive and unlabeled (PU) learning. Furthermore, in many of these applications, not only are the negative examples absent, the positive examples available for learning can also be rather limited. As such, several PU learning algorithms for time series classification have recently been developed to learn from a small set *P* of labeled seed positive examples augmented with a set *U* of unlabeled examples. The key to these algorithms is to accurately identify the likely positive and negative examples from *U*, but it has remained a challenge, especially for those uncertain examples located near the class boundary. This paper presents a novel ensemble based approach that restarts the detection phase several times to probabilistically label these uncertain examples more robustly so that a reliable classifier can be built from the limited positive training examples. Experimental results on time series data from different domains demonstrate that the new method outperforms existing state-of-the art methods significantly.

Keywords: Ensemble based system, positive and unlabeled learning, time series classification.

1 Introduction

Many real-world data mining application domains, such as aerospace, finance, manufacturing, multimedia and entertainment, involve time series classification [1-3]. For example, a typical aircraft health monitoring application in aerospace would be to classify the states of the airplane engines into either the normal or faulty states based on time series sensor readings from multiple sensors (e.g. vibration and temperature sensors) attached to the aircraft. Most of classification methods directly apply traditional supervised learning techniques that rely on large amounts of labeled examples from predefined classes for learning. In practice, this paradigm is not practical because collecting and labeling large sets of data for training are often very expensive if not impossible.

Researchers have proposed alternative learning techniques to build classifiers from a small amount of labeled training data enhanced by a larger set of unlabeled data that are typically easy to collect. These methods include semi-supervised learning [4-6] and Positive Unlabeled learning (PU learning) [7-13]. While both approaches exploit

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 243–257, 2012.
© Springer-Verlag Berlin Heidelberg 2012

the unlabeled data (*U*) to enhance the performance of their classifiers, they differ in their training data requirements: PU learning only requires positive data (*P*) whereas semi-supervised learning still requires both positive and negative training data. PU learning is therefore applicable in a wide range of application domains, such as text classification, medical informatics, pattern recognition, bioinformatics and recommendation system, where negative data are often unavailable. However, the applications of PU learning to classify time series data have been relatively less explored due to specific challenges of time series classification such as high feature correlation [14]. As far as we know, there are only 3 research works that applied PU learning approaches for time series data classification.

The pioneering work, proposed by Wei and Keogh [14], iteratively expands the positive set from the initial positive examples using the unlabeled data that are most similar to them in terms of Euclidean distance, with the remaining unlabeled data being extracted as negative data. The method is highly dependent on having a good stopping criterion; otherwise, early stopping will result in an expansion of only a small number of positives, with highly noisy negatives. To improve the algorithm, a more recent work [15] attempted to propose a good stopping criterion by using the historical distances (in this case, dynamic time warping distance) between candidate examples from *U* to the initial positive examples. Although the refinement has enabled more positive examples to be extracted, it is still unable to identify *accurate* positives (and hence negatives) from *U*, especially when the actual positives and negatives in *U* are severely unbalanced. The experimental results reported showed high precision but very low recall for classification.

More recently, to tackle the challenge of constructing accurate boundary between positive and negative data in *U*, we proposed a new PU approach called LCLC for time series classification [16]. Unlike the previous methods, LCLC adopts a cluster-based approach instead of instance-based approach. First, the unlabeled set *U* is partitioned into small unlabeled local clusters (*ULCs*) using the *K-means* algorithm [17]. All the examples within an individual cluster will be assigned a same label as either *LP* (Likely Positive) or *LN* (Likely Negative). The local clusters are also exploited for more robust feature selection for classification. A cluster chaining approach is then applied to extract the boundary positive and negative clusters (*ULCs*) to estimate the decision boundary between the actual positives and negatives in *U*. LCLC has been demonstrated to perform much better than the first two PU learning methods, as it can identify the boundary positive and negative clusters from *U* more accurately. While LCLC's cluster-based approach (i.e. all the instances within an individual cluster will be assigned the same label) is more robust than traditional instance-based approach, in practice, not all the examples within the individual local clusters will actually be from same class. This means that some instances within each cluster may be misclassified. When these misclassified examples (especially when they are in the boundary clusters) are used to build the final 1-NN classifier (i.e. classification based on the top one nearest neighbor; Keogh et al [18] performed a comprehensive empirical evaluation on the current state-of-the-arts which shows 1-NN to be the best technique), the performance of overall LCLC algorithm is less satisfactory than expected. In Section 2, we will go into the further details of the LCLC algorithm as well as the reason that LCLC algorithm generates false positives/negatives.

In this paper, we propose a novel ensemble based approach En-LCLC (Ensemble based Learning from Common Local Clusters) to overcome the drawbacks of the LCLC algorithm. Our proposed En-LCLC method adopts an ensemble-based strategy by performing the LCLC algorithm multiple times on different cluster settings to obtain multiple diverse classifiers. We can then assign each instance with a "soft" probabilistic confidence score based on its overall classification results that could better indicate each instance's class label. Based on the probabilistic scores, we also identify and remove potential noisy instances which could confuse our classifier. An Adaptive Fuzzy Nearest Neighbor (AFNN) classifier is then constructed based on the clean set of "softly labeled" positive and negative instances identified.

The rest of the paper is organized as follows. We provide an overview on the LCLC algorithm in Section 2. We then present our proposed En-LCLC algorithm in Section 3. Results from extensive experiments on time series data across diverse fields reported in Section 4 show that the classifiers built using En-LCLC algorithm can indeed identify the ground truth's positive and negative boundaries more accurately, leading to improvements in classification accuracy. Finally, Section 5 concludes the paper.

2 LCLC Algorithm and Its Weakness

In this section, we describe the LCLC method proposed in [16] in further details. As mentioned earlier, the first step of LCLC algorithm groups the unlabeled data *U* into local clusters and selects independent and relevant features based on these clusters. Subsequently, LCLC algorithm extracts reliable negative set *RN* from *U,* with the remaining clusters belonging to *U-RN* regarded as ambiguous clusters (*AMBI*). Finally, LCLC determines likely positive clusters *LP* and negative clusters *LN* from *AMBI* using cluster chaining, and the final classifier is built using all the extracted positives and negatives from *U*.

Algorithm 1 shows the main steps of the LCLC. Steps 1 and 2 perform the local clustering and feature selection. In Step 1, LCLC partitions the unlabeled data *U* into small local clusters ULC_i (i=1, 2, …, K) using *K-means* clustering method. Each local cluster ULC_i is then treated as an observed variable of the time series data, and it assumes that all the instances belonging to a local cluster share the same principal component and have the same class label.

In Step 2, the *Clever-Cluster* method [19] is then used to select *K* common feature subset from the positive set *P* and a partitioned coherent unlabeled clusters ULC_i. It first computes the principal components for each time series observations which are the positive set *P* and unlabeled clusters ULC_i. Descriptive common principal components are then computed across all these principal components and used to select *K* highest mutual information features. Interested readers could find more details in [16]. The intuition for such selection is based on the observation that a well-selected subset of the common principal features can capture the underlying characteristics of the time series dataset to enable accurate extraction of the remaining *hidden* positives/negatives from *U*.

Step 3 identifies the Reliable Negative set *RN* from *U* based on the similarities between the local clusters ULC_i to the initial positive cluster *P*. In this step, LCLC computes the Euclidean distance of each ULC_i from the positive set *P* using common principal features extracted in Step 2. After that, it extracts those local clusters which are farthest away from *P* and store them into *RN*. The size of *RN* is set to contain about a half of the local clusters, while the other half is considered as ambiguous clusters *AMBI* in Step 4.

Algorithm 1. LCLC algorithm

```
input: Initial positive data P, Unlabeled dataset U, number of clusters K
    1. K-ULCs ← Partition U into K local clusters using K-means;
    2. Select K features from the raw feature set ← Clever-Cluster(P, K-ULCs);
    3. Extract Reliable Negative Examples RN from the Unlabeled dataset U;
    4. Define the ambiguous clusters AMBI = U – RN;
    5. Identify likely positive clusters LP and likely negative clusters LN from the
       AMBI clusters using cluster chaining for boundary decision;
    6. Build a 1-NN classifier using P together with LP as a positive training set,
       and RN together with LN as a negative training set.
```

By now (after Step 4), LCLC algorithm has obtained a positive data *P* and reliable negative data *RN* that can be used to further extract the *likely* positive clusters *LP* and the *likely* negative clusters *LN* from the ambiguous clusters *AMBI* which are near the positive and negative boundary. Step 5 performs a novel *cluster chaining* method to label these boundary clusters. The basic idea of cluster chaining is to build cluster chains starting from the positive *P*, going through one or more *AMBI* clusters, and finally stopping at a reliable negative cluster in *RN*. Figure 1 illustrates the scenario where there are two reliable negative clusters *RN* far away from the positive cluster *P*, and 4 *AMBI* clusters located between the positive cluster (*P*) and negative clusters (*RN*). Two cluster chains have been built here. For each cluster chain, LCLC finds the *breaking link* (decision boundary) with *maximal distance* between the clusters that separates the cluster chain into two sub-chains. All the *AMBI* clusters within the sub-chain that contains *P* will be regarded as likely positive clusters and stored into *LP,* while the *AMBI* clusters within the other sub-chain that includes *RN* are regarded as likely negative clusters and stored into *LN*.

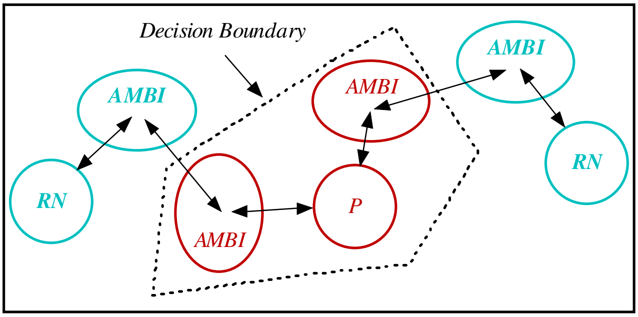


Fig. 1. Cluster chaining for boundary decision

Finally, LCLC uses *P* together with *LP* as a positive training set, and *RN* together with *LN* as a negative training set to build the final 1-NN classifier for time series classification.

Although LCLC works better than the existing PU learning methods identifying the boundary clusters more accurately, we observed that it still has two drawbacks. Firstly, it assumes that *all the instances belonging to a local cluster have the same class label*. Clustering ensures that *most* of the examples within a same cluster belong to the same class, but some of the examples within same cluster could belong to other classes. By assigning the same label to all the examples within each cluster, LCLC will misclassify some examples (typically minority class examples within individual clusters), ultimately affecting the performance of the classifier trained on these mis-assigned examples. The errors introduced will be especially costly for those examples located in the boundary clusters between positive and negative classes.

Secondly, as LCLC algorithm uses *K-means* algorithm to perform clustering, it will generate different clusters based on *K* randomly initialized centroids. It is highly possible that the examples near the positive and negative boundary will be grouped into different clusters and assigned with different labels each time we perform LCLC algorithm. This source of random errors can introduce further limitations in the overall performance of LCLC.

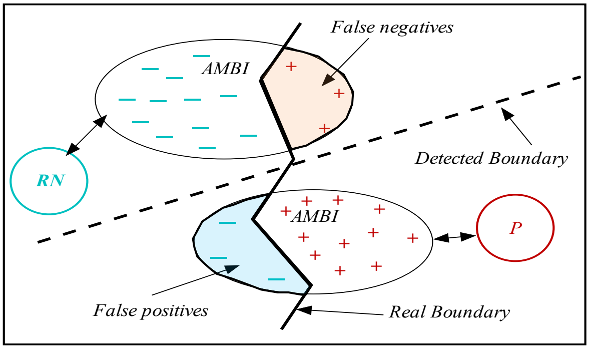


Fig. 2. Drawback of the LCLC on fixed clustering assignment to every instance

Figure 2 depicts the scenario of having possible misclassified instances in the local clusters near the real positive and negative boundaries. We can see that some instances may be clustered wrongly and some assigned with wrong labels based on its container cluster's label. The region represented by dashed green (orange) areas shows the set of false negative (positive) examples. The probability to be misclassified is high for those instances that are close to the class boundaries, leading to decreased accuracy of the final classification. This motivates us to propose a more robust approach to address the issue.

3 The Proposed Technique En-LCLC

In this section, we present our proposed En-LCLC algorithm (Ensemble based Learning from Common Local Clusters) to overcome the drawbacks in the original LCLC. We propose an ensemble based approach that restarts LCLC algorithm multiple times to assign labels for each instance in the unlabeled data. We then generate an integrated "soft" probabilistic label to the examples based on their classification results from diverse classifiers. Following that, we filter and remove uncertain instances. We then design an Adaptive Fuzzy Nearest Neighbor (AFNN) classifier to train on the enhanced dataset that have been assigned with clean soft labels to better reveal the ground truths' positive and negative boundaries.

3.1 Probabilistic Soft Labeling Using Diverse Classifiers

We make use of the randomness of clusters generated by *K-means* clustering used in the LCLC algorithm to create diverse classifiers. We perform *n* times LCLC algorithm in which each *K-means* clustering with random initialization will produce different cluster settings for constructing cluster chains and deciding the boundary clusters. Each time corresponds to a different classifier setting. We use the probability distribution of the *n* labels generated by *n* diverse classifiers for each example as its "soft" labels. By integrating them together probabilistically, we minimize the potential bias of individual LCLC prediction and the expected errors in the extracted likely positive/negative examples by our ensemble-based approach can be expected to be reduced.

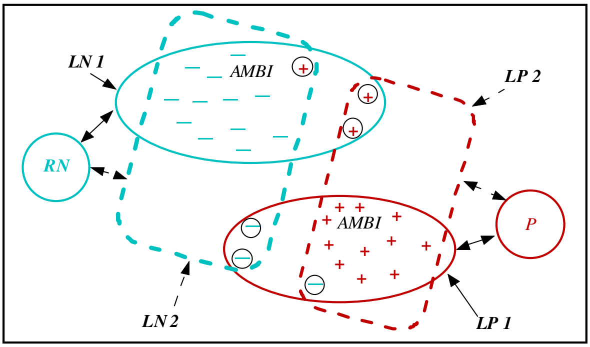


Fig. 3. An example of En-LCLC by repeating LCLC on two different clustering configurations

Figure 3 illustrates a scenario in which we have two classifiers with different cluster settings (i.e. *n*=2). In the first cluster setting for classifier 1, *K-means* clustering has generated two ambiguous clusters denoted by *LP1* and *LN1*. A cluster chain is constructed from *P* through *LP1*, *LN1*, and finishes at *RN*, and LCLC eventually determined all the examples in *LP1* as positive and all the examples in *LN1* as negative class. Note that there are three positive and three negative instances that are misclassified in this case. In the second cluster setting for classifier 2, *K-means* clustering generated two ambiguous clusters, denoted by *LP2* and *LN2*; and the corresponding cluster chain goes through *P, LP2*, *LN2*, and ends at *RN.* The eventual LCLC labels for the examples in *LP2* are positives and the examples in *LN2* are negatives. In this case, only 1 positive and 1 negative are misclassified. With a big *n*, we can integrate the results together probabilistically and give the misclassified examples low confidence scores as they will be given inconsistent labels using different classifiers.

Algorithm 2 shows the details to generate the confidence score for each example in the unlabeled set *U-P* (*AMBI* clusters and *RN* clusters). Note that we have given a confidence score of 1 for all the positive examples in *P*.

Algorithm 2. Confidence score generation

```
Input: one initial seed positive $s$, unlabeled dataset $U$, number of iterations $n$
1. Use Wei's method to get an initial positive set $P$;
2. Repeat step 3 to step 5 $n$ times
3.     Partition the remaining unlabeled data $U - P$ into $K$ unlabeled local
       clusters using K-means clustering with random initialization;
4.     Perform LCLC (cluster chain breaking) to extract the reliable
       negative clusters $RN$, likely positive clusters $LP$, likely negative
       clusters $LN$;
5.     Compute the weights for each instance using equation (1) and (3);
6. Compute the normalized confidence score for each instance using equation
   (4) and (5);
```

We need to cater for cases where only a small number of positives are available for learning, even in the extreme scenario of having only one seed positive example. As such, similar with LCLC, in the first step, we adopt Wei's method [14] for this task as follows. Given the positive seed *s*, we add the next most confident positive instances from *U* until the stopping criterion is reached, that is, when there is a drop of the *minimal nearest neighbor distance*. Wei's method uses this early stopping criterion because when a negative example is mistakenly added into *P*, there is a high chance that we will keep adding more negative examples, for the negative space is expected to be much denser than the positive space [14]. While this method tends to provide an early stop instead of proceeding to find the actual boundary between the positives and negatives, we observe that it is useful for constructing a robust positive set *P* with very high precision. In other words, we can obtain a "pure" positive set *P* which is still reasonably bigger than the original one seed positive example set to work with.

We repeat *n* times Steps 3 to 5 with different initializations for *K-means* clustering to create an ensemble system with *n* diverse classifiers. In Step 4, instead of assigning a "hard" label to all the instances within a cluster, we assign a "soft" probabilistic label to each instance according to its contribution to the container cluster. In particular, given an instance x_i, its probability P_{x_i} to be labeled as its belonging cluster C_{x_i} 's label is defined using Gaussian distribution as:

$$P_{x_i} = \exp\left(-\frac{1}{2} \frac{dist(centroid(C_{x_i}), x_i)^2}{\sigma^2} \right) \tag{1}$$

where *centroid* (C_{x_i}) is the cluster C_{x_i} 'centroid, σ is the *width* of the Gaussian distribution which is defined as the standard deviation of distances between all the instances in C_{x_i} to *centroid*(C_{x_i}), i.e.

$$\sigma = \sqrt{\frac{1}{\left|C_{x_i}\right|} \sum_{i=1}^{|C_{x_i}|} \left[dist\left(centroid(C_{x_i}), x_i\right) - mean(dist) \right]^2}, x_i \in C_{x_i} \tag{2}$$

The probability P_{x_i} denotes the possibility that an instance x_i has its container cluster's label. If it is near its cluster centroid *centroid*(C_{x_i}), then it has a higher probability to belong to cluster's label; otherwise, it will be given a lower probability.

The probability is converted as a weight for each instance depending on whether it is extracted into the positive or negative class:

$$w_{x_i}^j = \begin{cases} P_{x_i} & \text{if its container cluster is } P \text{ or } LP \\ -P_{x_i} & \text{if its container cluster is } LN \text{ or } RN \end{cases} \tag{3}$$

Since we perform LCLC n times, we use j ($1 \le j \le n$) to indicate that the weight is given in j-th iteration of En-LCLC algorithm.

Step 6 computes the confidence score that an instance is extracted as either positive or negative, which is simply defined as follows:

$$\lambda_i = \sum_{j=1}^{n} w_{x_i}^j \tag{4}$$

Equation (4) basically sums all the weights over n iterations into a consolidated score. An instance will get a bigger positive (negative) value if it is extracted as a positive (negative) consistently. On the other hand, if it is assigned a near zero value (no matter positive or negative), it is an unreliable instance which may not be very useful for further classification step.

We compute the confidence score of each instance normalized to the range of [-1,1] as follows:

$$\lambda_{nor,i} = \frac{\lambda_i - \frac{1}{2}(\max(\lambda_i) + \min(\lambda_i))}{\frac{1}{2}(\max(\lambda_i) - \min(\lambda_i))} \tag{5}$$

Each instance x_i in *U-P* will have a confidence score $\lambda_{nor,i}$ that indicates its propensity to be a positive or negative instance. The instances in *U-P* (all with normalized confidence scores) together with *P* (all with a fixed confidence score of 1) may serve as the training set *TRAIN* for learning a classifier.

Note that the training data *TRAIN* may still contain some uncertain instances which have low confidence scores. Before building our final classifier, we perform noise-filtering pass [20] to further remove the possibly incorrectly labeled instances. To do so, for each example x_i, we define its adaptive neighborhood $N(x_i)$ which consists of its *minimal* number of nearest neighbors whose total confidence score is larger or equal to 1 (which indicates we have enough information from the neighbors to make accurate decision). If $sign(\lambda_{nor,i}) \neq sign(\sum_{j=1}^{|N(x_i)|} \lambda_{nor,j})$, or the instance will be misclassified by its nearest neighbors within its adaptive neighborhood (label inconsistent), then we will remove the confusing instance from our training set *TRAIN.*

Algorithm 3. Adaptive Noise-filtering procedure

Input: Training set *TRAIN* with normalized confidence score $\lambda_{nor,j}$ for each instance in *TRAIN*

```
1.  Sort the instances in TRAIN by their normalized confidence score in the
    decreasing order;
2.  For all $x_i \in$ TRAIN do
3.      $K=1$;
4.      Find the nearest neighbor $x_j$, and add it to $N(x_i)$;
5.      While $\sum_{j=1}^{K} |\lambda_{nor,j}| < 1$
6.          $K=K+1$;
7.          Find Kth nearest neighbor $x_j$ and add it to $N(x_i)$;
8.      If $sign(\lambda_{nor,i}) \neq sign(\sum_{j=1}^{|N(x_i)|} \lambda_{nor,j})$
9.          $removalflag(x_i)=1$;
10. For all $x_i \in$ TRAIN do
11.     If ($removalflag(x_i) == 1$)
12.         $TRAIN = TRAIN - \{ x_i \}$;
```

Algorithm 3 shows the detailed step of this noise filtering process. For each example in training set, Steps 3-7 find its adaptive neighborhoods and Steps 8-9 detect if it is the noisy instance. Finally, we remove all these noisy instances from the training set in Steps 10-12.

3.2 Combining Classifiers Using Adaptive Fuzzy Nearest Neighbor Method

Traditional time series data classification often employed the k-nearest neighbor (KNN, especially 1-NN) method to build final classifier based on the given "hard" labeled training data [18]. Our approach is able to generate more refined "soft" labeled training data with confidence scores. Instead of applying a fixed threshold (which is difficult to determine for arbitrary datasets) on the confidence scores to provide "hard" labeling of the training data to enable employing the conventional KNN approaches, we will build an adaptive fuzzy nearest neighbor classifier as it can effectively make use of the normalized confidence scores of the training instances.

The detailed algorithm for building our Adaptive Fuzzy Nearest Neighbor (AFNN) classifier is shown in Algorithm 4. Steps 1-10 classify all the test instances based on their prediction values (Step 7), which are the accumulated normalized confidence scores of the nearest neighbors from the adaptive neighborhood. Steps 8-10 classify a test instance based on the sign of its prediction value.

Algorithm 4. Adaptive Fuzzy Nearest Neighbor
Input: Training set *TRAIN*, Test set *TEST*, $\lambda_{nor,j}$ for each instance x_i in *TRAIN*

```
1.  For all $x_i \in TEST$ do
2.      $K = 1$;
3.      Find the nearest neighbor $x_j$, $x_j \in TRAIN$;
4.      While $\sum_{j=1}^{K} |\lambda_{nor,j}| < 1$
5.          $K = K + 1$;
6.          Find $K$th nearest neighbor $x_j$, $x_j \in TRAIN$;
7.      Compute the prediction value: $Pre(x_i) = \sum_{j=1}^{K} \lambda_{nor,j}$;
8.      If $Pre(x_i) \geq 0$
9.          Label $x_i$ as positive;
10.     Else    Label $x_i$ as negative.
```

4 Empirical Evaluation

We compare our proposed technique En-LCLC algorithm against three existing state-of-the-art PU learning methods for time series classification, namely, Wei's method [14], Ratanamahatana's method (denoted as Ratana's method) [15] as well as the LCLC method [16].

4.1 Experimental Data, Settings and Evaluation Metric

Similar to the experiments reported in [14] and [16], we have performed our empirical evaluation on the five diverse time series datasets across different fields from [21] and the UCR Time Series Data Mining archive [22] to facilitate comparison. The details of the datasets are shown in Table 1.

Table 1. Datasets used in the evaluation experiments

Name	Training set		Testing set		Num of Features
	Positive	Negative	Positive	Negative	
ECG	208	602	312	904	86
Word Spotting	109	796	109	796	272
Wafer	381	3201	381	3201	152
Yoga	156	150	156	150	428
CBF	155	310	155	310	128

In our empirical evaluation, we repeated the experiments performed in [14] and [16] by randomly selecting just one seed example from the positive training class for the learning phase (i.e. seed *s* in Algorithm 2), with the rest of the training set (both positives and negatives) treated as unlabeled data (ignoring their labels; *U* in Algorithm 2).

We repeat our experiments 10 times with different initial seed positive instances for each dataset and report the average values of the 10 results. Since our proposed En-LCLC performs *n* diverse classifiers to generate the confidence scores for each instance, we set $n = 15$ for this work. We will also evaluate *n*'s sensitivity later.

We use the F-measure to evaluate the performance of the four PU learning techniques. The F-measure is the harmonic mean of precision (*p*) and recall (*r*), and it is defined as $F=2*p*r/(p+r)$. In other words, the F-measure reflects an average effect of both precision and recall. F-measure is large only when both precision and recall are good. This is suitable for our purpose to accurately classify the positive and negative time series data. Having either too small a precision or too small a recall is unacceptable and would be reflected by a low F-measure.

4.2 Experimental Results

Table 2 shows the overall classification results of all the four techniques. The results showed that both LCLC [16] and En-LCLC performed much better than the other two earlier methods for time series classification, namely, Wei's method [14], and Ratana's method [15]. Our proposed En-LCLC produced the best classification results across all the 5 datasets, achieving F-measures of 86.9%, 76.2%, 77.5%, 89.1% and 81.6%, which are 0.2%, 3.5%, 5.1%, 3.7% and 11.5% higher than the second best results from LCLC. On average (last row), En-LCLC was able to achieve 82.3% F-measure, which is 4.8% higher than the second best LCLC method in terms of F-measure, indicating that En-LCLC is indeed well-designed for time series data classification.

Table 2. Overall performance of various techniques

Dataset	Wei's method	Ratana's method	LCLC	En-LCLC
ECG	0.405	0.840	0.867	**0.869**
Word Spotting	0.279	0.637	0.727	**0.762**
Wafer	0.433	0.080	0.724	**0.775**
Yoga	0.466	0.626	0.854	**0.891**
CBF	0.201	0.309	0.701	**0.816**
Average	0.357	0.498	0.775	**0.823**

Recall that En-LCLC has two key steps for our classification task: (1) using "soft" Adaptive Fuzzy Nearest Neighbor classifier replace the "hard" label 1-NN classifier, and (2) performing a noise filtering before building our final classifier. To determine their individual effects on our classification performance, we also test the En-LCLC algorithm without these key steps. The results are shown in Table 3.

Table 3. En-LCLC with 1-NN and without noise filtering

Dataset	ECG	Word Spotting	Wafer	Yoga	CBF
En-LCLC w 1-NN	0.867	0.736	0.745	0.861	0.754
En-LCLC w/o noise filtering	0.868	0.749	0.764	0.879	0.792
En-LCLC	**0.869**	**0.762**	**0.775**	**0.891**	**0.816**

Without using “soft” AFNN classifier, we use 1-NN by converting the confidence scores into “hard” labels using the following procedure: if the confidence score of an unlabeled instance is larger than 0, then it is regarded as a positive training example; otherwise, it is treated as a negative training example. We observe from Table 3 that En-LCLC with 1-NN is on average 1.8% worse than our proposed En-LCLC. It is important to note that En-LCLC with 1-NN has already benefited from the more accurate confidence scores obtained from our ensemble-based strategy. Similarly, by including noise filtering, En-LCLC is able to perform 1.22% higher on average, indicating that removing those potentially noisy examples using the confidence scores can contribute to enhance the classification performance.

Table 4 compares the performance of the LCLC and En-LCLC techniques for extracting positives and negatives from unlabeled data. In the table, *Ext_P* (*Ext_N*) represents the number of positives (negatives) extracted. *Error_P* and *Error_N* represent the error rate of positive and negative extraction respectively. Compared with LCLC, En-LCLC made 0.40%, 6.80%, 5.60%, 1.90% and 1.90% less errors for positive extraction over the 5 datasets. It also made 0.10%, 1.10%, 0.50%, 4.10% and 5.90% less errors for negative extraction over 5 datasets. As these instances are located near the positive and negative boundary, the reduction in false positives and false negatives helped improve En-LCLC’s eventual accuracy.

Table 4. Extraction comparison between LCLC and En-LCLC

Dataset		ECG	Word Spotting	Wafer	Yoga	CBF
LCLC	*Ext_P*	234.3	139.4	269.7	166.8	106.6
	Error_P	16.1%	36.3%	16.6%	17.3%	12%
	Ext_N	575.7	765.6	3312.3	139.2	358.4
	Error_N	2%	2.6%	4.7%	12.9%	17.1%
En-LCLC	*Ext_P*	233	134.8	271.4	165.7	118.6
	Error_P	15.7%	29.5%	11%	15.4%	10.1%
	Ext_N	574.3	745.6	3299.8	134.8	321
	Error_N	1.9%	1.5%	4.2%	8.8%	11.2%

We now analyze the efficiency of En-LCLC algorithm. The main steps for our algorithm include *K-means* clustering to partition the unlabeled data, compute the confidence score for each example, perform noise filtering, and construct Adaptive Fuzzy Nearest Neighbor (AFNN) classifier. These steps can be performed using efficient algorithms implemented in linear time. Note that our ensemble-based approach will not increase the overall time complexity although it performs *n* times classification (*n* is typically a relatively small integer). Our proposed algorithm is therefore not less efficient than the other methods. Figure 4 shows that our proposed En-LCLC method can scale well over all the datasets (we implemented our En-LCLC algorithm using Matlab and running on the Desktop Intel Pentium IV core 2 Duo 3.0 GHz CPU using Microsoft Windows XP with 4.0 GB memory).

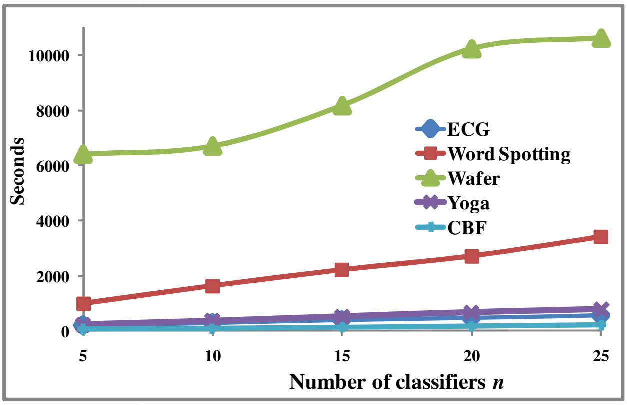


Fig. 4. Efficiency analysis of the number of classifiers *n*

Finally, we evaluate the effect of parameter *n*, the number of classifiers, which is the only one parameter of our method. In this paper, we have set *n* = 15. We repeated our experiments with different values of *n* from 5 to 25, with a step of 5. Figure 5 shows the sensitivity of En-LCLC with different values of *n*. We can see that with the increasing value of *n* from small values, the F-measure also increased for 4 datasets (interestingly, the parameter *n* does not affect the performance of ECG data much). When *n*>= 15, the performance has reached steady status, without much changes, suggesting that we do not need a very large *n* to benefit from ensemble based strategy.

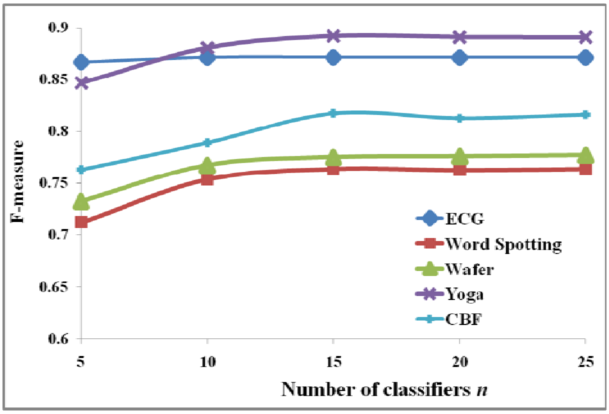


Fig. 5. Sensitivity analysis of the number of classifiers *n*

5 Conclusions

Time series classification has been applied in many real-world applications across different domains. In this paper, we study the positive unlabeled learning method as it eliminates the tedious and costly process to hand-label large amounts of training data. We proposed a novel En-LCLC algorithm (Ensemble based Learning from Common Local Clusters) to overcome the drawbacks of the existing state-of-the-art LCLC algorithm. Our proposed En-LCLC algorithm adopts an ensemble-based strategy which performs LCLC algorithm multiple times to minimize the potential bias of individual LCLC prediction. As shown in our experiments, the error rates in the extracted likely positive/negative examples has been effectively reduced. We designed an Adaptive Fuzzy Nearest Neighbor classifier to exploit the “soft” confidence scores obtained from the diverse classifiers. We also made use of the confidence scores to remove unreliable examples from the training dataset. The experimental results show that our proposed method performed much better than existing methods over multiple time series data from different domains.

References

[1] Olszewski, R.T.: Generalized Feature Extraction for Structural Pattern Recognition in Time-Series Data, PhD thesis, Carnegie Mellon University, Pittsburgh, PA (2001)

[2] Rath, T.M., Manmatha, R.: Word image matching using dynamic time warping. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2, pp. II-521–II-527 (2003)

[3] Xi, X., Keogh, E., Shelton, C., Wei, L., Ratanamahatana, C.A.: Fast time series classification using numerosity reduction. In: Proceedings of the 23rd International Conference on Machine Learning. ACM, Pittsburgh (2006)

[4] Chapelle, O., Scholkopf, B., Zien, A.: Semi-Supervised Learning. MIT Press (2006) (in Press)

[5] Li, M., Zhou, Z.-H.: SETRED: Self-Training with Editing. In: Ho, T.-B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 611–621. Springer, Heidelberg (2005)

[6] Zhu, X.: Semi-supervised learning literature survey, Technical report, no.1530, Computer Sciences, University of Wisconsin-Madison (2008)

[7] Liu, T., Du, X., Xu, Y., Li, M.-H., Wang, X.: Partially Supervised Text Classification with Multi-Level Examples. In: AAAI (2011)

[8] Gabriel Pui Cheong, F., Yu, J.X., Hongjun, L., Yu, P.S.: Text classification without negative examples revisit. IEEE Transactions on Knowledge and Data Engineering 18, 6–20 (2006)

[9] Li, X., Liu, B.: Learning to classify texts using positive and unlabeled data. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence. Morgan Kaufmann Publishers Inc., Acapulco (2003)

[10] Li, X., Liu, B., Ng, S.-K.: Learning to Identify Unexpected Instances in the Test Set. In: Proceedings of Twentieth International Joint Conference on Artificial Intelligence, India (IJCAI 2007), pp. 2802–2807 (2007)

[11] Li, X., Yu, P., Liu, B., Ng, S.-K.: Positive Unlabeled Learning for Data Stream Classification. In: SDM, pp. 257–268 (2009)

[12] Liu, B., Lee, W.S., Yu, P.S., Li, X.: Partially Supervised Classification of Text Documents. In: ICML (2002)

[13] Elkan, C., Noto, K.: Learning Classifiers from Only Positive and Unlabeled Data. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2008)

[14] Wei, L., Keogh, E.: Semi-supervised time series classification. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, Philadelphia (2006)

[15] Ratanamahatana, C., Wanichsan, D.: Stopping Criterion Selection for Efficient Semi-supervised Time Series Classification. In: Lee, R. (ed.) Soft. Eng., Arti. Intel., Net. & Para./Distri. Comp. SCI, vol. 149, pp. 1–14. Springer, Heidelberg (2008)

[16] Nguyen, M.N., Li, X., Ng, S.-K.: Positive Unlabeled Learning for Time Series Classification. In: Proceedings of International Joint Conference on Artificial Intelligence, IJCAI (2011)

[17] Kanungo, T., Mount, D.M., Netanyahu, N.S., Piatko, C.D., Silverman, R., Wu, A.Y.: An efficient k-means clustering algorithm: analysis and implementation. IEEE Transactions on Pattern Analysis and Machine Intelligence 24, 881–892 (2002)

[18] Keogh, E., Kasetty, S.: On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. Data Mining and Knowledge Discovery 7, 349–371 (2003)

[19] Yoon, H., Yang, K., Shahabi, C.: Feature subset selection and feature ranking for multivariate time series. IEEE Transactions on Knowledge and Data Engineering 17, 1186–1198 (2005)

[20] Wilson, D.L.: Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. IEEE Transactions on Systems, Man and Cybernetics 2, 408–421 (1972)

[21] Wei, L.: Self Training dataset (2007), http://alumni.cs.ucr.edu/~wli/selfTraining/

[22] Keogh, E.: The UCR Time Series Classification/Clustering Homepage (2008), http://www.cs.ucr.edu/~eamonn/time_series_data/

Efficient Mining Regularly Frequent Patterns in Transactional Databases

Md. Mamunur Rashid[1], Md. Rezaul Karim[1], Byeong-Soo Jeong[1], and Ho-Jin Choi[2]

[1] Dept. of Computer Engineering, Kyung Hee University
1 Seochun-dong, Kiheung-gu, Yongin-si, Kyunggi-do 446-701, Republic of Korea
[2] Computer Science Dept., Korea Advanced Institute of Science and Technology
335 Guseong-dong, Yuseong-gu, Daejeon 305-701, Republic of Korea
{mamun,asif_karim,jeong}@khu.ac.kr, hojinc@kaist.ac.kr

Abstract. Finding interesting patterns plays an important role in several data mining applications, such as market basket analysis, medical data analysis, and others. The occurrence frequency of patterns has been regarded as an important criterion for measuring interestingness of a pattern in several applications. However, temporal regularity of patterns can be considered as another important measure for some applications. In this paper, we propose an efficient approach for miming regularly frequent patterns. As for temporal regularity measure, we use variance of interval time between pattern occurrences. To find regularly frequent patterns, we utilize pattern-growth approach according to user given *min_support* and *max_variance* threshold. Extensive performance study shows that our approach is time and memory efficient in finding regularly frequent patterns.

Keywords: Data mining, interesting pattern, frequent pattern, regularly frequent pattern.

1 Introduction

Mining patterns that appear frequently in transactional database has been actively and widely studied in data mining and knowledge discovery techniques such as association rule mining, classification, clustering, time-series mining, graph mining, and web mining [1, 2, 9]. The common framework behind frequent pattern mining is that only patterns occurring at a high frequency are interest to users. Recently, researchers have focused on devising methods to mine user interest-based frequent patterns to produce the desired result set in efficient manner by applying early pruning technique to reduce the size of resultant itemset. In literature several interesting parameters such as closed [4, 5], maximum [6], K-most [7], and demand-driven [8] has been proposed for finding frequent patterns.

Another important criterion for identifying the interestingness of frequent patterns might be the shape of occurrence, i.e., whether they occur regularly, irregularly, or mostly in specific time interval in the database. For example, in a retail market,

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 258–271, 2012.
© Springer-Verlag Berlin Heidelberg 2012

among all frequently sold products, the user may be interested only on the regularly (e.g., ‘drinks’, ‘tissues’) sold products compared to other products that are frequently sold in specific time duration (e.g., ‘electric fans’ during the summer, ‘electric heaters’ during the winter and ‘rain-coats’ during the rainy season). Similarly, for improved web administration an administrator may be interested on the click sequences of heavily hit web pages. In genetic data analysis these to fall genes that not only appear frequently but also co-occur regular interval in DNA sequence may carry more effective information to scientists. Among the heavy network traffics, the regular network access patterns may provide significant information in network monitoring. In telecommunications and analysis of sensor network data, finding the occurrence regularity can be very necessary in other application such as moving object detection. Form the above examples; we can see that user may be interested on the appearance characteristics (regularity) of frequent patterns. Hence, regularity plays an important role in finding interesting frequent patterns for a wide verity of applications. We define such a frequent pattern that occurs after regular intervals in a database as regularly frequent patterns.

Let us consider the database in Table 1 with 9 transactions. It can be examined that the supports of the patterns “A”, “C”, “AB”, “BC” and “BE” in the database are respectively 5, 5, 4, 4 and 4.Though these patterns may be frequent in the database some of them may not be regularly frequent patterns because non-regular appearance intervals. For example “A”, “AB” and “BE” occurs more frequently at a certain part of the database (i.e., “A” at the beginning and “BE” at the middle of database) then the rest part. On the other hand, patterns “C” and “BC” etc are appear within after regular intervals throughout the database. So, they can be more important frequent patterns in terms of regular occurs intervals. The typical frequent pattern mining methods fail to find such regularly frequent patterns because they are only concerned about the support and not consider the pattern appearance behavior.

Motivated by the above discussion and examples, in this paper, we introduce the problem of finding regularly occurs frequent patterns in a transactional database. We define a new regularity measure for a pattern by the variance of intervals at which the same pattern occurs in a database. For regularly frequent patterns mining, we use a tree structure, called a RF-tree (Regularly Frequent Pattern tree), which capture the database contents in a highly compact manner. To ensure that the tree structure is compact and informative, only frequent length-1 items will have nodes in the tree and more frequently occurring items are located at the upper part of the tree to have better chance of prefix sharing. We use a pattern growth approach to mine the regularly frequent patterns from our RF-tree. Our performance study shows that the proposed approach is efficient in finding regularly frequent patterns.

This paper is organized as follows. Section 2, we describe background. In Section 3, we explain our proposed approach for regularly frequent pattern mining problem. In Section 4, we represent the details of the structure of RF-tree and the regularly frequent pattern mining process. In Section 5, we represent experimental results. Finally, in Section 6, conclusions are presented.

Table 1. A transactional database

ID	Transaction	ID	Transaction	ID	Transaction
1	A D	4	A B E F	7	C D E
2	A B E F	5	A B C E	8	D E F
3	A B C E	6	B C D	9	B C D

2 Background

At first, Agrawal et al. [1], proposed a support constraint-based technique to mine the frequent patterns. They used the *downward closure* property to prune the infrequent patterns. This property states that if a pattern is infrequent then all of its super patterns must be infrequent. But this technique suffers from candidate generation-and-test problem and needs several database scans. Han et al. [2] proposed the frequent pattern tree (FP-tree) and FP-growth algorithm which overcome the problem of candidate generation-and-test of [1] and needs only two-database scans to find all the frequent patterns.

The number of occurrences may not always represent the significance of a pattern. Mining interesting patterns from database plays an important role in data mining research. Mining periodic patterns [10, 11, 12, 13] and cyclic patterns [13, 14] in a static database have been well-addressed over the last decade. Periodic pattern mining problem in time series data focuses on the cyclic behavior of patterns either full periodic pattern mining [11, 13] and partial periodic pattern mining [12] of time series. Recently, Tanbeer et al. [3] proposed the RP (Regular Pattern tree) to mine the regularly occurs patterns. The basic model of regular patterns is as follows [3].

Let $I = \{i_1, i_2, ..., i_n\}$ be a set of items. A set $X \subseteq I$ is called a pattern (or an itemset). A pattern containing ‘k’ number of items is called k-itemset. A transaction t = (tid, Y) is a tuple where tid is the transaction-id and Y is a pattern. A transactional database T is a set of transactions $T = \{t_l, ..., t_m\}$ with m = |T|, i.e., total number of transactions in database. If $X \subseteq I$, it is said that X occurs in t and such tid is denoted as t_j^X, $j \in [1,m]$.Let $T^X = \{t_j^X, ..., t_k^X\} \subseteq T$ where $j \le k$ and $j, k \in [1, m]$ be the ordered set of transactions in which pattern X has occurred. Let t_s^X and t_t^X, where $j \le s < t \le k$ be the two consecutive transactions inT^X. The number of transactions or time difference between t_s^X and t_t^X, can be defined as a period of X, say p^X. Then a period of X, $p^X = t_s^X - t_t^X$.Let $P^X = \{p_1^X, ..., p_s^X\}$ be the set of periods for pattern X. For simplicity, assume the first and last transactions in database as ‘null’ with $t_f = 0$ and $t_l = t_m$ respectively. The regularity of X, denoted as $Reg(X) = \max(p_1^X, ..., p_s^X)$. The pattern X is called regular if $Reg(X) \le maxPrd$ where *maxPrd* is the user-specified maximum regularity constraints.. The regularity of a pattern can be described in percentage of |T|.

Example 1. Consider the transactional database shown in Table 1. tid is a identification of each transaction in this database. The set of items in the database, I={A,B,C,D,E}. For instance, in Table 1 the set of transactions where pattern “AB” appears in tids 2, 3, 4, and 5. So T^{AB}={ 2, 3, 4, 5}. The periods for this pattern are 2 (= 2- t_f), 1 (= 3 - 2), 1(= 4 - 3), 1 (= 5 - 4) and 4 (= t_l - 4), where t_f = 0 represents the initial transaction and t_l = 9 represents the last transaction in database. The regularity of ab, Reg(AB)=max(2, 1, 1, 1, 4) = 4. If the user-specified *maxPrd = 3*, then “AB” is not a regular pattern because *Reg(AB)>maxPrd.*

But in many real-world applications, it is difficult for the patterns to appear regularly without any interruptions. So in erroneous or noisy environment, *maxPrd* measure for regularity calculation is not effective.

Example 2. The set of all transactions where pattern “C” appears in the database of Table 1 is T^C= {2, 5, 6, 7, 9}. Then the period of pattern “C” is P^C= {2, 3, 1, 1, 2, 0}. For a value of *maxPrd=3*, pattern “C” is a regular pattern. But if the item “C” deleted anyhow from tid=5 for error or noise or if it not appear in id =5, then the pattern become irregular for given maxPrd. Although the pattern “C” occurs regular manner at the end of the database but only one large interval makes it irregular.

Even though mining regular patterns are closely related to our work, but we cannot directly applied it for finding regularly frequent patterns from transactional database because regular patterns do not consider the support threshold which is only constraint to be satisfied by all frequent patterns. On the other hand, our proposed regularly frequent pattern mining technique, introduce a new interesting measure of regularity based on variance of intervals and provide the set of patterns that satisfy support thresholds and regularity threshold finally in a transactional database.

3 Proposed Model

To mine frequent pattern that occur regularly in a database, we propose a new methodology. For temporal regularity measure, we use variance of interval time between pattern occurrences instead of *maxPrd.* We use the notions period and set of periods that are defined in section 2.

Definition 1 (Regularity of Patten X): Let for a T^X, P^X be the set of all periods of X i.e., $P^X = \{ p_1^X, p_2^X, ..., p_n^X \}$, where n is the total number of periods in P^X. Then the average period value of pattern X represent as,

$$\bar{X} = \sum_{k=1}^{N} \frac{p_k^X}{n} \tag{1}$$

Then the variance of periods for pattern X is represent as,

$$\sigma^X = \sum_{k=1}^{N} \frac{(p_k^X - \bar{X})^2}{n} \tag{2}$$

The regularity of X can be denoted as *Reg(X)* = σ^X (variance of periods for pattern X).

Example 3. In the database of Table 1, pattern "BF" occurs in tid=3 and tid=5. Then T^{BF} = {3, 5} and P^{BF} = {3, 2, 4}. Then average period value for pattern "BF"=9/3=3 and variance value, $\sigma^{BF} = \frac{(3-3)^2+(2-3)^2+(4-3)^2}{3} = 0.67$

The maximum period of the pattern "BF" is 4. So, If the user-specified *maxPrd=3*, then "BF" is not a regular pattern according the exits method, because *Reg(BF)>maxPrd.* On the other hand, if the user-specified variance value of interval, *max_variance = 1.0*, then pattern BF is a regular pattern, because the variance of interval of "BF" is 0.67 i.e. *Reg(BF) < max_variance.*

From above example, we can see that only one large interval can makes a pattern irregular if we use *maxPrd* as regularity. On the other hand, if we use variance of interval time between pattern occurrences in database, then one large interval of that pattern has small effect in the value of variance calculation. Therefore, by using variance of interval as regularity we can mine regular pattern properly.

Definition 2 (Support): The number of transactions in a database T that contain X is called the support of X in database T and is denoted as *Sup(X)* = $|T^X|$, where $|T^X|$ is the size of T^X. For example the support of pattern "BF" in the database T of Table 1 is *Sup(BF)=2*, since $|T^{BF}|=2$.

Definition 3 (Regularly frequent Pattern): A pattern is called a regularly frequent pattern if it satisfies both of the following two conditions: (i) its support is no less than a user-given minimum support threshold, say, *min-sup*, α and (ii) its regularity is no greater than a user-given maximum regularity threshold say, *max_variance*, β.

Problem Definition: Given a transactional database T, *min-sup* (α) and *max_variance* (β) constraints, the objective is to discover the complete set of regularly frequents in T having than support no less than α and regularity no more β.

4 RF-tree: Design, Construction and Mining

In this section, we first introduce RF-tree (Regularly Frequent Pattern tree) for mining regularly frequent patterns and then discuss the efficient mining technique on it. To facilitate high degree of compactness in the tree structure, items in a RF-tree are arranged in frequency-descending item order. A frequency-descending tree can provide not only a highly compact tree structure but also an effective mining process using pattern growth mining techniques. This has shown and proven in [2] and [9]. Our given regularity (variance of internals) does not maintain downward closure property. Since only the frequent items will play important role in the frequent-pattern mining, so it is need to perform one database scan to identify the set of frequent length-1 items say, F for a given *min_sup*. At this scan, as we know the each length-1 items tid occurrence information, therefore, we can easily calculate the regularity by definition 1. To facilitate the tree traversal and to store all length-1 items, like FP-tree [2], an item header list, called frequent list (F-list) is built. This list consists of each distinct item with relative support (i.e., frequency).Once the F-list is built, we generate F by removing items that are not frequent. The support values of items in F are then used for sorting the F-list in support descending order to facilitate the RF-tree construction. In the next sections, we describe the structure and construction of RF-tree.

4.1 Structure of RF-tree

The structure of a RF-tree consists of one root node referred to as the "null", a set of item-prefix sub-trees (children of the root), and a frequent item list, called the F-list. Similar to an FP-tree, each node in a RF-tree represents an itemset in the path from the root up to that node. An important feature of a RF-tree is that, in the tree structure it maintains the appearance information for each transaction. To explicitly track such information, it keeps a list of tid information only at the last item-node for a transaction. Hence, a RF-tree maintains two types of nodes; say ordinary node and tail node. The former are types of nodes used in FP-tree that do not maintain tid information. On the other hand, the latter type used in RP-tree [3], can be defined as follows:

Definition 4 (tail node): Let t={$y_{1,} y_{2,...,} y_n$} be a transaction that is sorted according to the F-list order. If t is inserted into RF-tree in this order, then the node of the tree that represents item y_n is defined as the tail-node for t and it explicitly maintains t's tid. Irrespective of the node type, no node in RF-tree needs to maintain a
support count value like FP-tree. Each node in the RP-tree maintains parents, children, and node traversal pointers. So, the structures of an ordinary node and a tail node are given as follows:

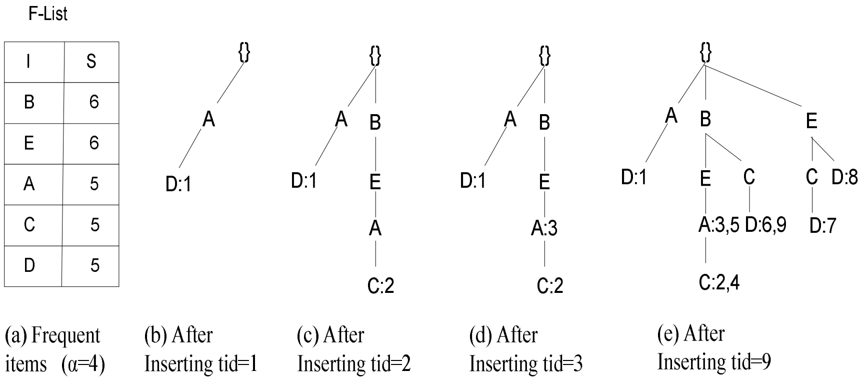
For ordinary node: M, where M is the item name of the node.

For tail node: M [$t_{1,} t_{2,...,} t_n$], where M is the item name of the node and t_i, $i \in$ [1,n], is a transaction-id in the tid-list, indicating that M is the tail-node for transaction t_i.

From above definitions and the RF-tree node structure we can deduce the following lemma.

Lemma 1: A tail-node in an RF-tree inherits an ordinary node; but not vice versa.

Proof: The structure of an ordinary node states that it exactly maintains three types of pointers: a parent pointer, a list of child pointers, and a node traversal pointer. A tail-node maintains all such information like an ordinary node. It also maintains the tid-list, which is additional information. Since the tid-list is not maintained in an ordinary node, so we can say, there is an ordinary node in every tail-node and in contrast, no tail-node in an ordinary node.


F-List

I	S
B	6
E	6
A	5
C	5
D	5

(a) Frequent items (α=4) (b) After Inserting tid=1 (c) After Inserting tid=2 (d) After Inserting tid=3 (e) After Inserting tid=9

Fig. 1. Construction of RF-tree

4.2 Construction of RF-tree

The RF-tree is constructed in such a way by the second scan of database that, it only contains nodes for items in F-list and is arranged in sorted F-list order. The construction of the RF-tree is similar to that of the FP-tree but the node structure of the RF-tree significantly differs from the FP-tree. We use the following example to illustrate the step-by-step construction process of the RF-tree in Fig. 1.

Consider the database of Table 1. At the first scan of database derives a list of F items, {(A:5), (B:6), (C:5), (D:5), (E:6), (F:3)} (the number after ":" indicates the support of item). Fig. 1 (a) shows the frequency-descending F-list for *min_sup=4*, which is constructed through the first database scan by removing the items with *min_sup<4*. Then, item "F" is removed because its *min_sup<4*. Then only the items in the F-list of Fig. 1 (a) are involved in RF-tree construction that follows the FP-tree [2] construction technique to insert any sorted transaction into the tree. To simplify the figures we do not show the node traversal pointers in trees.

The tree constructing starts with inserting the first transaction {A, D} according to F-list order, as shown in Fig. 1 (b). Since all items in the transactions are frequent, they are listed in the RF-tree. The node "D:1"is the tail-node for the transaction. Hence, it carries the tid (i.e., 1) of the transaction in its tid-list. The scan of the second transaction leads to the construction of the second branch of the RF-tree as in Fig. 1 (c). We insert {A, B, C, E} in the order of {B, E, A, C}in the tree with node "C:2" being the tail-node that carries the tid information for the transaction. Then we insert the third transaction {A, B, E, F} in the tree. Since only the frequent items are inserted in the tree. After removing the infrequent items from tid=3 (i.e., 'F') and sorting the remaining frequent items according to the F-list, we insert {A, B, E, F} in the order of {B, E, A} in the tree which shares a common prefix (B, E, A) with the existing path (B, E, A, C) and created ("A:3") as a tail node with value 3 in its tid-list is shown in Fig. 1 (d). After inserting all the transactions in similar fashion the final RF-tree we get is shown in Fig.1. (e).

Property 1: An RF-tree contains a complete set of frequent item projection for each transaction in TDB only once.

One can assume that the structure of RF-tree may not be memory efficient, since it explicitly maintains tids for each transaction in the tree. Like [3], we argue that, RF-tree achieves the memory efficiency by keeping only transaction information at the tail node and avoiding the support count field at each node. In the best case, when every transaction is same, then the number of tail node being one. On the other hand, in worst case, when every transaction is different, then number of tail-nodes is equal to the number of transactions in database.

4.3 Mining Regularly Frequent Patterns

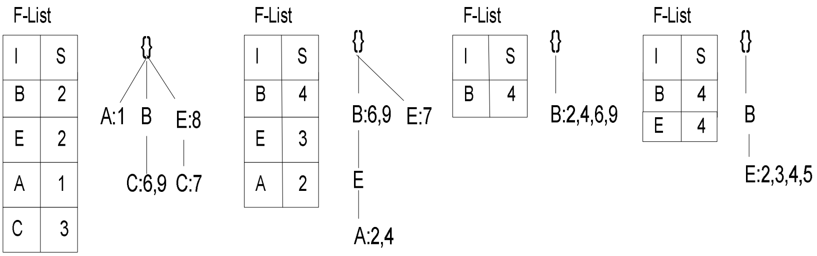
Construction of a highly compact RF-tree enables the subsequent mining of the regularly frequent patterns by using a pattern growth approach. Similar to the FP-growth [2] mining approach, we recursively mine the RF-tree of decreasing size to generated regularly frequent patterns by creating conditional pattern-bases (PB) and corresponding conditional trees (CT) without additional database scan. Though both RF-tree and FP-tree arrange items support-descending order, we cannot directly apply the FP-growth mining on a RF-tree, because RF-tree does not maintain the frequency count at each node. Moreover it needs to handle the tid-lists at tail-nodes during mining. For this reason, we devise a pattern growth mining technique that can handle the additional features of the RF-tree.

For regularly frequent patterns mining from RF-tree, the basic operations are (i) counting length-1 frequent items (ii) constructing a conditional pattern-base for each regular item and (iii) constructing the conditional tree for each conditional pattern-base. Then generate the frequent pattern from the conditional tree. At last we check the regularity of generated frequent pattern to find regularly frequent pattern. Before discussing these operations in details, we explore the following important property and lemma of an RF-tree like RP-tree [3].

Property 2: The tid-list in a RF-tree maintains the occurrence information for all the nodes in the path (from that tail-node to the root) at least in the transactions of the list.

Lemma 2: Let X={b_1, b_2…, b_n} be a path in a RF-tree where node b_n is the tail node that carries the tid-list of the path. If the tid-list is pushed-up to node b_{n-1}, then the node b_{n-1} maintain the occurrence information of the path X'={ b_1, b_2…, b_{n-1} } for the same set of transactions in tid-list without any loss.

Proof: Based on Property 2, the tid-list at node b_n maintains the occurrence information of the path Z' at least in transactions it contains. So, the same tid-list at node b_{n-1} exactly maintains the same transaction information for Z' without any lose.



F-List

I	S
B	2
E	2
A	1
C	3

(a) Conditional pattern-base for 'D'

F-List

I	S
B	4
E	3
A	2

F-List

I	S
B	4

(b) Conditional pattern-base and conditional tree for 'C'

F-List

I	S
B	4
E	4

(c) Conditional pattern-base and conditional tree for 'A'

Fig. 2. Conditional pattern-base and conditional tree construction with the RF-tree

F-list provides the length-1 frequent items. The conditional pattern-base is constructed starting from the item at the bottom of the F-list. At the time of creating the conditional pattern-base, a small F-list for that item is also created. For our example, "D" is the bottom-most item in the F-list. The conditional pattern-base tree of D is shown in Fig. 2 (a). According to the lemma 2, the tid-list of D is pushed-up to its respective parent nodes A, C, C and E. So, each parent node of D is converted to a tail-node. For node D, its immediate frequent pattern is (D:1, 6, 7, 8, 9) i.e., D occurs in transaction 1,6,7,8 and 9 so its support is 5 and it has four paths in RF-tree: (A:1, D:1), (B, C, D:6, 9), (E, C, D:7) and (E, D:8) where number after ":" indicates each sub-pattern occurring tid. Since D's conditional pattern-base {(A:1), (B, C:6,9), (E, C:7), (E:8)} generates no frequent items, the mining of D is terminates and only frequent-1 pattern is D. Then we check the regularity of pattern D. Since we know the occur transactions of D, then we easily calculate the regularity of D by definition 1, which is 2.54. In this example, we set our regularity threshold, *max_variance=1.0.* Since Reg(D)>1.0, therefore, D is not regularly frequent pattern.

For node C, its immediate frequent pattern is (C:2, 4, 6, 7 ,9) and it has three path, (B, E, A, C:2, 4), (B, C:6, 9) and (E, C:7). Then C's conditional pattern-base is {(B, E, A:2, 4), (B:6, 9), (E:7)} which is shown in Fig. 2 (b). Then C's conditional tree leads only one branch (B:2, 4, 6, 9) and generated frequent patterns are (BC:2, 4, 6, 9) and (C:2, 4, 6, 7, 9). Then, we calculate the regularity of BC and C by definition 1 and get their respective regularity values are 0.96 and 0.583. Since {Reg(BC), Reg(C)}<1.0, so BC and C both are regularly frequent patterns.

For node A, its immediate frequent pattern is (A:1, 2, 3, 4, 5) and it's conditional pattern-base is (B, E:2, 3, 4, 5), which is shown in Fig. 2 (c). Then A's conditional tree leads only one branch (B, E:2, 3, 4, 5) and generated frequent patterns are (AB:2, 3, 4, 5), (AE:2, 3, 4, 5), (ABE:2, 3, 4, 5), and (A:1, 2, 3, 4, 5). If we calculate the regularity of these patterns, we get their respective values are 1.36, 1.36, 1.36 and 1.25. Since, {Reg(AB), Reg(AE), Reg(ABE), Reg(A)}>1.0, these patterns are not regularly frequent patterns.

Similarly, node E derives (E:2, 3, 4, 5, 7, 8) and its conditional pattern-base is (B:2, 3, 4, 5). E' conditional tree leads only once branch (B:2, 3, 4, 5) and generated frequent patterns are (BE:2, 3, 4, 5) and (E:2, 3, 4, 5, 7, 8) and their respective regularity is 1.23 and 0.204. Since Reg(BE)>1.0, so, BE is not regularly frequent and Reg(E)<1.0, therefore E is a regularly frequent pattern.

Table 2. Mining the RF-tree by creating conditional (sub-) pattern-bases

Item	Conditional pattern-base	Conditional tree	Frequent pattern	Regularly frequent pattern
D	{(A:1),(B,C:6,9), (E,C:7), (E:8)}	Ø	(D:1,6,7,8,9)	Ø
C	{(B,E,A:2,4), (B:6,9), (E:7)}	(B:2,4,6,9)	(BC:2,4,6,9), (C:2,4,6,7,9)	BC,C
A	(B,E:2,3,4,5)	{(B:2,3,4,5), (E:2,3,4,5)}	(AB:2,3,4,5), (AE:2,3,4,5), (ABE:2,3,4,5), (A:1,2,3,4,5)	Ø
E	(B:2,3,4,5)	(B:2,3,4,5)	(BE:2,3,4,5, (E:2,3,4,5,7,8)	E
B	Ø	Ø	(B:2,3,4,5,6,9)	B

Node B derives only (B:2, 3, 4, 5, 6, 9) but no conditional pattern-base. If we calculate the regularity of B, we get 0.77 which is less than our given regularity threshold, so B is regularly frequent pattern. The conditional pattern-base, the conditional-trees, generated frequent patterns and regularly generated frequent patterns are summarized in Table 2.

From the Table 2, we can see that for our example database in Table 1, there are 10 frequent patterns but the numbers of regularly frequent pattern are only 4.

With the above mining process, one can see that, from an RF-tree constructed on a TDB, the complete set of regularly frequent patterns for given *min_sup* and *max_variance* thresholds with the pattern growth approach. This approach is efficient due to the support descending item order in the RF-tree structure.

5 Experimental Results

In this section, we present the experimental results on mining regularly frequent patterns on RF-tree. Our programs are written in Microsoft Visual C++ and run with Windows XP on a 2.66 GHz machine with 1GB of main memory. To evaluate the performance of our proposed approach, we have performed experiments on IBM synthetic dataset (T10I4D100K) and real life dataset musroom from frequent itemset mining dataset repository (http://fimi.cs.helsinki.fi/data/).

Our experimental analysis divided into three parts. First, we study the compactness of RF-tree; second, we show its performance with mining the set of regularly frequent patterns; and finally, we give the results to prove the scalability in mining regularly frequent patterns.

Table 3. Memory comparison between RF-tree and FP-tree

Dataset min_sup, α (%)	Tree	max_variance, β (%)	Memory (MB)		
			α_1	α_2	α_3
T10I4D100K α_1=1.0, α_2=2.0, α_3=3.0	RF-tree	0.15	0.45	0.415	0.321
		0.30	5.750	3.825	0.386
		0.45	7.675	4.250	0.390
	FP-tree	--	8.350	4.655	0.412
musroom α_1=10, α_2=20, α_3=30	RF-tree	1.5	0.127	0.125	0.110
		3.0	0.178	0.138	0.115
		4.5	0.232	0.140	0.117
	FP-tree	--	0.285	0.175	0.115

5.1 Compactness of the RF-tree

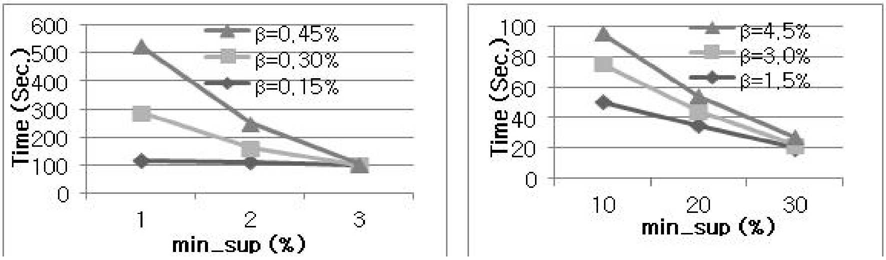
To show the compactness of RF-tee we compared its size with FP-tree [2] for given support and regularity thresholds. For FP-tree, we considered min_sup, because it is a support threshold-based tree structure. The memory consumptions of RF-tree and FP-tree on the change of *min_sup* (α) and *max_variance* (β) values over synthetic dataset (*T10I4D100K*) and real life dataset *musroom* are shown in Table 3. The first column of the table shows the datasets and several *min_sup* (α) and the changes of *max_variance* (β) are presented in column 3.

From Table 3, we can see that, keeping the min_sup fixed the memory usages of RF-tree increases with the increase of *max_variance* values. On the other hand, for keeping the *max_variance* fixed the tree size become smaller with increasing values of *min_sup*. From Table 3, we also see that, even though the tree size increases with the decrease and increase of *min_sup* and *max_variance* respectively, a RF-tree achieves compactness similar to or better than an FP-tree for most *min_sup* and *max _variance* values.

5.2 Execution Time of the RF-tree

In the second experiment, we show the effectiveness of RF-tree in mining regularly frequent pattern mining in terms of execution time. FP-tree [2] store only the frequent items with a given threshold and RP-tree [3] is designed only the regularity threshold.

Therefore, it is not possible to find the set of regularly frequently patterns from FP-tree and RP-tree. In this experiment we show the performance variation of only RF-tree with the changes of thresholds. The performance on execution time experiments for the given datasets is shown in Fig. 3. For the given *min_sup* and *max_variance* thresholds, the execution time is shown in the y-axis of the graphs.



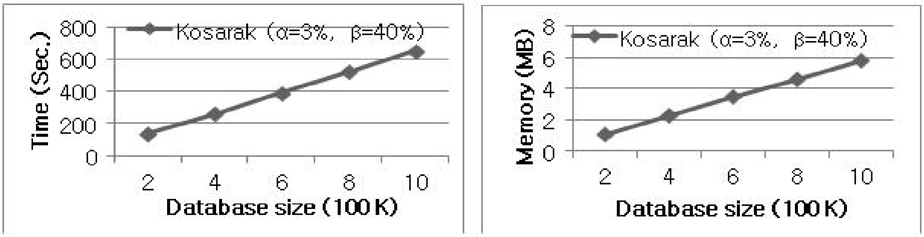
(a) Execution time over T10I4D100K (b) Execution time over musroom

Fig. 3. Execution time over RF-tree

The x-axis in each graph shows the change of *min_sup* value in the form of percentage of database size. It is shown from the Fig. 3 that, for sparse dataset *T10I4D100K* and dense dataset *musroom*, RF-tree takes almost similar amount of time for the variations of the *max_variance* values when *min_sup* values are relatively high. On the other hand, as the *max-variance* go down, the gaps become wider.

5.3 Scalability of the RF-tree

We study the scalability of the RF-tree by varying the number of transactions in the database on execution time and required memory. For this test we use *kosarak* dataset for its huge sparse dataset with a large number of distinct items (41,270) and transactions (990,002). This dataset is divided into five portions each of 0.2 million transactions. The experimental results are present in Fig. 4, where we fix *min_sup* 3% and *max_variance* 40%. Fig. 4 (a) shows the total execution time including the RF-tree constructing time and corresponding mining time in y-axis and the number of transactions are put in the x-axis. Fig. 4 (b) shows the required memory in y-axis with the increase of database size. From Fig. 4, we can see that RF-tree shows stable result of about linear increase of execution time and memory usage with respect of the size of the database.



(a) Over execution (b) Over Memory

Fig. 4. Scalability of RF-tree

6 Conclusions

In this paper, we have introduced an interesting pattern mining problem, called regularly frequent pattern mining from transactional database under the user given support and regularity thresholds. The effectiveness of finding such patterns in real-world applications are also shows in this paper. We use RF-tree, a highly compact tree structure to capture the database content, and a pattern growth mining approach to finding the complete set of regularly frequent patterns. The experimental result shows that RF-tree can provide the time and memory efficiency during mining regularly frequent patterns.

Acknowledgement. This work was supported by a grant from the NIPA (National IT Industry Promotion Agency) in 2012. (Fostering Global IT Human Resource Project) and the National Research Foundation (NRF) grant (No. 2011-0018264) of Ministry of Education, Science and Technology (MEST) of Korea.

References

1. Agrawal, R., Imielinski, T., Swami, A.N.: Mining Association Rules Between Sets of Items in Large Databases. In: ACM SIGMOD International Conference on Management of Data, pp. 207–216 (1993)
2. Han, J., Pei, J., Yin, Y.: Mining Frequent Patterns without Candidate Generation. In: ACM International Conference on Management of Data, pp. 1–12 (2000)
3. Tanbeer, S.K., Ahmed, C.F., Jeong, B.-S., Lee, Y.-K.: RP-Tree: A Tree Structure to Discover Regular Patterns in Transactional Database. In: Fyfe, C., Kim, D., Lee, S.-Y., Yin, H. (eds.) IDEAL 2008. LNCS, vol. 5326, pp. 193–200. Springer, Heidelberg (2008)
4. Chi, Y., Wang, H., Yu, P.S., Muntz, R.R.: Catch the Moment: Maintaining Closed Frequent Itemsets Over a Data Stream Sliding Window. Knowledge and Information System 10(3), 265–294 (2006)
5. Zaki, M.J., Hsiao, C.-J.: Efficient algorithms for mining closed itemsets and their lattice structure. IEEE Transactions on Knowledge and Data Engineering 17(4), 462–478 (2005)

6. Huang, Y., Xiong, H., Wu, W., Deng, P., Zhang, Z.: Mining maximal hyperclique pattern: A hybrid serach stategy. Information Sciences 177, 703–721 (2007)
7. Minh, Q.T., Oyanagi, S., Yamazaki, K.: Mining the K-Most Interesting Frequent Patterns Sequentially. In: Corchado, E., Yin, H., Botti, V., Fyfe, C. (eds.) IDEAL 2006. LNCS, vol. 4224, pp. 620–628. Springer, Heidelberg (2006)
8. Wang, H., Perng, C.-S., Ma, S., Yu, P.S.: Demand-driven frequent itemset mining using pattern structures. Knowledge and Information Systems 8, 82–102 (2005)
9. Tanbeer, S.K., Ahmed, C.F., Jeong, B.-S., Lee, Y.-K.: Efficient Single-Pass Frequent Pattern Mining using a prefix-tree. Information Sciences 179, 559–583 (2009)
10. Tanbeer, S.K., Ahmed, C.F., Jeong, B.-S., Lee, Y.-K.: Discovering Periodic-Frequent Patterns in Transactional Databases. In: Theeramunkong, T., Kijsirikul, B., Cercone, N., Ho, T.-B. (eds.) PAKDD 2009. LNCS (LNAI), vol. 5476, pp. 242–253. Springer, Heidelberg (2009)
11. Elfeky, M.G., Aref, W.G., Elmagarmid, A.K.: Periodicity detection in time series databases. IEEE Transactions on Knowledge and Data Engineering 17(7), 875–887 (2005)
12. Han, J., Dong, G., Yin, Y.: Efficient mining of partial periodic patterns in time series database. In: Proceedings of 15th International Conference on Data Engineering, pp. 106–115 (1999)
13. Ozden, B., Ramaswamy, S., Silberschatz, A.: Cyclic Association Rules. In: 14th International Conference on Data Engineering, pp. 412–421 (1998)
14. Toroslu, I.H., Kantarcıoǧlu, M.: Mining Cyclically Repeated Patterns. In: Kambayashi, Y., Winiwarter, W., Arikawa, M. (eds.) DaWaK 2001. LNCS, vol. 2114, pp. 83–92. Springer, Heidelberg (2001)

Fast Tree-Based Mining of Frequent Itemsets from Uncertain Data

Carson Kai-Sang Leung* and Syed K. Tanbeer

Department of Computer Science, University of Manitoba, Canada
kleung@cs.umanitoba.ca

Abstract. Over the past two decades, numerous algorithms have been proposed for mining frequent itemsets from *precise* data. However, there are situations in which data are *uncertain*. In recent years, tree-based algorithms have been proposed to mine frequent itemsets from uncertain data. While the key success of tree-based algorithms for mining precise data is due to the compactness of a tree structure in capturing precise data, the corresponding tree structure in capturing uncertain data may not be so compact. In this paper, we propose a novel tree structure for capturing uncertain data such that it is as compact as the tree for capturing precise data. Moreover, we also propose two fast algorithms that use this compact tree structure to mine frequent itemsets. Experimental results showed the compactness of our tree and the effectiveness of our algorithms in mining frequent itemsets from uncertain data.

Keywords: Data mining, tree-based mining, frequent patterns, probabilistic databases, uncertain data.

1 Introduction and Related Work

Over the past two decades, there have been numerous studies [2,7,9,10] on mining frequent itemsets from *precise* data such as databases of market basket transactions, web logs, and click streams. In these databases of precise data, users definitely know whether an item is present in, or is absent from, a transaction in the databases. However, there are situations in which users are uncertain about the presence or absence of some items or events [3,4,11]. For example, a physician may highly suspect (but cannot guarantee) that a patient suffers from flu. The uncertainty of such suspicion can be expressed in terms of *existential probability*. So, in this probabilistic database of patient records, each transaction t_j represents a patient's visit to the physician's office. Each item within t_j represents a potential disease, and is associated with an existential probability value expressing the likelihood of a patient having that disease in t_j. For instance, in t_j, the patient has an 80% likelihood of having the flu, and a 60% likelihood of having a cold regardless of having the flu or not. With this notion, each item

* Corresponding author.

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 272–287, 2012.
© Springer-Verlag Berlin Heidelberg 2012

in a transaction t_j in traditional databases containing precise data can be viewed as an item with a 100% likelihood of being present in t_j.

As there are many real-life situations in which data are uncertain, efficient algorithms for mining uncertain data are in demand. To find frequent itemsets from *uncertain* data, several algorithms [5,6,12,14,16] have been proposed over the past few years, which include tree-based algorithms *UF-growth* [13] and *UFP-growth* [1]. Observing that FP-growth [8] (a tree-based algorithm) usually outperforms Apriori [2] when mining precise data, we previously proposed **UF-growth** [13] for mining uncertain data. Note that key success of FP-growth over Apriori is mainly due to its FP-tree, which is a compact tree structure capturing frequent items within transactions in the databases of precise data. By extracting appropriate tree paths to construct subsequent FP-trees, frequent itemsets can be mined. Each tree path represents a transaction. Each node in a tree path captures (i) an item x and (ii) its actual support (i.e., occurrence count of x in that tree path). Tree paths (from the root) are merged if they share the same items (i.e., the captured transactions share the same prefix items). Due to this path sharing, the FP-tree is usually compact. However, when dealing with uncertain data, the situation is different (because each item is associated with an existential probability value). The *expected support* of any itemset X is the *sum of products* of existential probability of items within X. Hence, UF-growth [13] uses a *UF-tree* to capture frequent items within transactions of uncertain data. Each node in an UF-tree captures (i) an item x, (ii) its existential probability value, and (iii) the occurrence count of x in that tree path. By doing so, UF-growth finds *all and only* those frequent itemsets by computing the expected support of an itemset X (as the sum of products of the captured existential probability values). Tree paths are merged if they share the same items *and* existential probability values. Consequently, UF-trees may not be as compact as FP-trees.

To reduce the tree size, **UFP-growth** [1] groups similar nodes (i.e., nodes with the same item x but *similar* existential probability values) into a cluster. Each cluster of the item x captures (i) the maximum existential probability value of all nodes within the cluster and (ii) the number of existential probability values in each cluster. Depending on the clustering parameter, the resulting tree—namely, *UFP-tree*—may be as large as the UF-tree (i.e., no reduction in tree size). On the other hand, if the UFP-tree is smaller than the UF-tree, then UFP-growth may return approximate results (e.g., with false positives—infrequent itemsets).

In this paper, we study the following questions: Can we further reduce the tree size (say, smaller than UF-trees) while minimizing the number of false positives? How to mine frequent itemsets from the resulting tree? Our **key contributions** of this paper are as follows:

1. a *capped uncertain frequent pattern tree* (*CUF-tree*), which can be as compact as FP-trees; and
2. two mining algorithms, called *CUF-growth* and *CUF-growth**, which are guaranteed to find all frequent itemsets (i.e., *no* false negatives) from uncertain data.

The remainder of this paper is organized as follows. The next section provides background. We propose our CUF-growth and CUF-growth* algorithms in Sections 3 and 4, respectively. Experimental results are shown in Section 5. Section 6 gives conclusions.

2 Background

Let Item=$\{x_1, x_2, \ldots, x_m\}$ be a set of domain items and X=$\{x_1, x_2, \ldots, x_k\}$ be a k-itemset (i.e., a set of k items), where $X \subseteq$ Item and $k \in [1, m]$. A transaction database DB=$\{t_1, t_2, \ldots, t_n\}$ contains a collection of n transactions, and each transaction t_j is a subset of Item.

Unlike precise databases, each item x_q in a transaction t_j in probabilistic databases of uncertain data is associated with an **existential probability value** $P(x_q, t_j)$, which expresses the likelihood of x_q to appear in t_j. Note that $0 < P(x_q, t_j) \leq 1$.

When items in a k-itemset X in a transaction t_j (where $X \subseteq t_j$) are independent, the **expected support** or **existential probability** $P(X, t_j)$ of X in t_j is the product of the corresponding existential probability values of the items within X, i.e., $P(X, t_j) = \prod_{x \in X} P(x, t_j)$. To a further extent, the **expected support** $expSup(X)$ of X in DB can be computed by summing the expected support of X in t_j over all transactions t_j's in DB, i.e., $expSup(X) = \sum_{j=1}^{n} P(X, t_j) = \sum_{j=1}^{n} \prod_{x \in X} P(x, t_j)$, where $n = |DB|$.

An itemset X is **frequent** in a probabilistic database DB of uncertain data if its expected support $expSup(X)$ is no less than a user-specified minimum support threshold (denoted as *minsup*). Given a *minsup* and DB, **frequent itemset mining from uncertain data** is to find from DB all frequent itemsets having expected support higher than or equal to *minsup*.

3 Our CUF-growth Algorithm and CUF-tree Structure

In this section, we propose (i) the **CUF-tree structure** and (ii) the **CUF-growth algorithm** that uses CUF-trees to mine frequent itemsets from uncertain data.

First, let us introduce some terms that are relevant to the remainder of this paper. Note that the CUF-tree is constructed by considering an upper bound of existential probability for each transaction. We call this upper bound the **cap of the transaction existential probability**, as defined below.

Definition 1. *The* ***transaction cap*** *of a transaction t_j, denoted as $P^{Cap}(t_j)$, is defined as the product of the two highest existential probability values of items within t_j. Let h=$|t_j|$ represent the length of t_j, $M_1 = \max_{q \in [1,h]} P(x_q, t_j)$ and $M_2 = \max_{r \in [1,h], r \neq q} P(x_r, t_j)$. Then, $P^{Cap}(t_j) = \begin{cases} M_1 \times M_2 \text{ if } h > 1, \\ P(x_1, t_j) \text{ if } h = 1. \end{cases}$*

The transaction cap provides users with an upper bound of existential probability values of all possible k-itemsets (where $k > 1$) in each transaction, as stated in the following theorem. (Due to space limitation, we omit the proof.)

Theorem 1. *The existential probability of any k-itemset X (with $k \geq 1$) in a transaction t_j is always less than or equal to the transaction cap of t_j, i.e., $P(X, t_j) \leq P^{Cap}(t_j)$, where $X \subseteq t_j$.*

Definition 2. *The* ***cap of expected support*** *of an itemset X, denoted as $expSup^{Cap}(X)$, is defined as the sum of all* transaction caps *of t_j in which X occurs. In other words, $expSup^{Cap}(X) = \sum_{j=1}^{n}(P^{Cap}(t_j) \mid X \subseteq t_j)$, $n = |DB|$.*

Theorem 2. *Given any k-itemset X in DB (where $k \geq 1$), $expSup(X) \leq expSup^{Cap}(X)$.*

Based on Theorem 2, we obtain following conditions with respect to the cap of existential probability of X and the user-specified *minsup*: Given any k-itemset X (where $k \geq 1$) in DB...
(i) if $expSup^{Cap}(X) < minsup$, then X cannot be frequent;
(ii) if X is a frequent itemset, then $expSup^{Cap}(X)$ must be at least *minsup*.
Based on these two conditions, our pattern pruning technique guarantees that no false negatives would be in our mining result.

The downward closure property [2] of support has been identified as an important property to remarkably reduce the search space and achieve time efficiency during the frequent itemset mining process. According to this property, if an itemset is frequent, then all of its subsets are guaranteed to be frequent. Equivalently, if an itemset is infrequent, then none of its supersets can be frequent. The use of any parameter satisfying this property can be considered as an efficient technique for mining. Note that the proposed cap of expected support of an itemset (Definition 2) satisfies the downward closure property because the cap of expected support of any subset Y of an itemset X must be greater than or equal to that of X: $expSup^{Cap}(Y) \geq expSup^{Cap}(X)$ for $Y \subseteq X$. Based on the downward closure property of the cap of expected support, we then construct our CUF-tree and use it to mine all frequent itemsets in our CUF-growth algorithm.

3.1 Construction of the CUF-tree

In general, the existential probability value of an item may vary from one transaction to another. For example, the existential probability values of item c in transactions t_1, t_2, t_3, and t_4 in DB in Table 1 are 0.6, 0.5, 0.9, and 0.2, respectively. This implies that c is more likely to be in t_3 than in t_4. Such phenomenon

Table 1. A transaction database DB (*minsup*=1.0)

tID	Contents	Contents (after 1st scan)	P^{Cap}
t_1	$\{a$:0.6, b:0.9, c:0.6, e:0.6$\}$	$\{a$:0.6, b:0.9, c:0.6, e:0.6$\}$	0.54
t_2	$\{a$:0.6, b:0.5, c:0.5, d:0.7$\}$	$\{a$:0.6, b:0.5, c:0.5$\}$	0.30
t_3	$\{a$:0.2, c:0.9, e:0.4$\}$	$\{a$:0.2, c:0.9, e:0.4$\}$	0.36
t_4	$\{a$:0.9, c:0.2, d:0.2, e:0.8$\}$	$\{a$:0.9, c:0.2, e:0.8$\}$	0.72

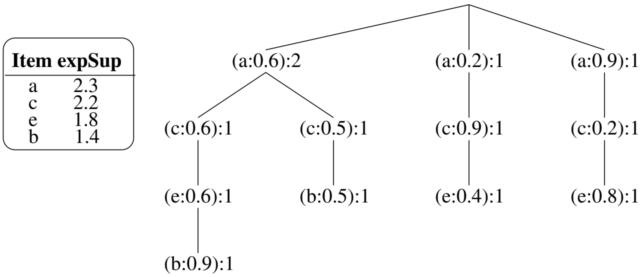
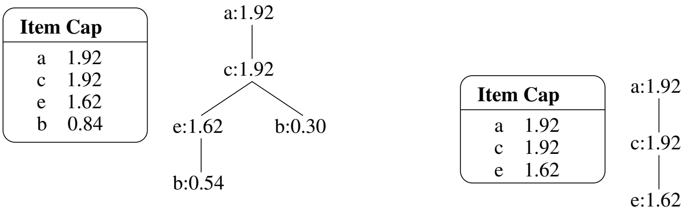


Fig. 1. The UF-tree for *DB* in Table 1 when using *minsup*=1.0



(a) Initial CUF-tree **(b)** CUF-tree w/ infrequent items removed

Fig. 2. CUF-trees for *DB* in Table 1 when using *minsup*=1.0

is not uncommon when dealing with probabilistic databases. When using the UF-tree [13], one needs to keep four nodes for c because the existential probability values for c are different in the four transactions in DB. Nodes can only be merged if they capture the same item and the same existential probability values. Fig. 1 shows the UF-tree constructed for DB when using *minsup*=1.0. Note that there are three a nodes as children of the root node. One cannot merge them because they do not share the same existential probability values. Hence, the UF-tree for capturing uncertain data may be large.

Understanding the above limitation, we propose a **Capped UF-tree (CUF-tree)**, which efficiently captures uncertain data and avoids repetition of the same node (even with different existential probability values). Instead of having different nodes for each existential probability value of the item, we store only a *transaction cap* value for each item. This makes the CUF-tree to be more compact. So, each node in our CUF-tree consists of (i) an item x, and (ii) the transaction cap (i.e., the sum of all transaction caps for those transactions that pass through or end at the node). See Fig. 2 (cf. Fig. 1).

Algorithm 1 shows our **CUF-tree construction algorithm**, which can be described as follows. The CUF-tree is constructed in two database scans. With the first scan of the database, we calculate the expected support of each domain item, remove infrequent items, and then sort all frequent items in descending order of their total expected support (lines 3 and 4). Then, the CUF-tree is constructed with the second database scan in a similar fashion as the construction

Algorithm 1: Construction of the CUF-tree

```
   Input: A transactional database $DB$, a user-specified minimum support threshold $minsup$
   Output: A CUF-tree capturing items from $DB$ with $minsup$
1  begin
2      $P^{Cap} = 0$, $h = 0$;
3      scan $DB$, calculate expected support and frequency of each item;
4      remove all infrequent items, arrange frequent items in a pre-defined order, let I-list be
       the resultant item list;
5      create the root node $R$ (= NULL) of a CUF-tree;
6      for each transaction $t_j \in DB$ do
7          delete all infrequent items from $t_j$;
8          if $t_j \neq$ NULL then
9              sort $t_j$ according to the order in I-list $h = |t_j|$;
10             if $h = 1$ then $P^{Cap} = P(x, t_j)$;
11             else $P^{Cap}$ = product of two highest existential probabilities of items in $t_j$;
12             call Insert_CUF-tree($R$, $t_j$, $P^{Cap}$);

13     for each item $x \in$ I-list do
14         if $expSup^{Cap}(x) < minsup$ then
15             delete $x$ and all of its nodes from the I-list and the CUF-tree, respectively;

16 Procedure Insert_CUF-tree($R$, $t_j$, $t^{Cap}$)
17 begin
18     let the sorted items in $t_j$ be $[x|X]$, where $x$ is the first item and $X$ is the remaining
       items in the list;
19     if a child $C$ of $R$ such that $C.item = x.item$ then
20         select $C$ as the current node; $C.t^{Cap} = C.t^{Cap} + t^{Cap}$;
21     else
22         create a new node $C$ as a child of $R$; $C.t^{Cap} = t^{Cap}$;
23     call Insert_CUF-tree($C$, $X$, $t^{Cap}$);
```

of FP-trees. When scanning each transaction in the second database scan, we compute its transaction cap (lines 10 and 11), insert items into the CUF-tree according to the sorted list order, and add the transaction cap value to each node (line 20). For better understanding of this CUF-tree construction process, let us consider the following example.

Example 1. Consider DB in Table 1, and let the user-specified support threshold $minsup$ be set to 1.0. The CUF-tree construction algorithm first scans DB once to construct an item list (or I-list for short) of expected support of each item in the same fashion as does in UF-tree by computing their expected support (i.e., a:2.3, b:1.4, c:2.2, d:0.9, e:1.8), removing infrequent items (e.g., d) from the list, and sorting the remaining items. This results in I-list=$\langle a$:2.3, c:2.2, e:1.8, b:1.4$\rangle$.

Each transaction in DB is then scanned the second time to compute its transaction cap and to construct the CUF-tree at the same time. As infrequent items have no contribution in frequent itemset mining, they are not inserted into the CUF-tree. When calculating the cap, we take in account only the frequent items in that transaction. This leads to a tighter upper bound (i.e., transaction cap) for each transaction to be maintained in the CUF-tree. For instance, even though the existential probability value of item d in t_2 (i.e., 0.7) is the maximum among all existential probability values in t_2 (i.e., $\max\{0.6, 0.5, 0.5, 0.7\}$), it does not contribute to the cap calculation for t_2 because d is an infrequent item. Hence, $P^{Cap}(t_2)$=0.6 × 0.5 = 0.3 (instead of 0.7 × 0.6 = 0.42), which eventually helps us to generate minimum number of false positives. The last column of Table 1 shows the transaction cap for each transaction in DB.

After computing the transaction cap of a transaction, we insert its frequent items into the CUF-tree according to *I-list* order. At the same time, we add its cap in the *I-list* to calculate the total cap of expected support of each of its items. Thus, transaction t_1 is inserted into the CUF-tree in the order of $\langle a, c, e, b \rangle$ with transaction cap 0.9×0.6 = 0.54 at each node. Transaction t_2 is then inserted in the same fashion. Since t_1 and t_2 share a common prefix (i.e., $\langle a, c \rangle$), we increase the transaction cap value for the nodes in the common prefix part by the value of $P^{Cap}(t_2)$ (i.e., 0.54+0.3). Nodes in the remaining part of t_2 carry the value of $P^{Cap}(t_2)$=0.3. Fig. 2(a) shows the contents of the CUF-tree after capturing all the transactions in DB in Table 1.

We can easily observe from the figure that the *I-list* of the CUF-tree maintains $expSup^{Cap}(X)$ for each 1-itemset X. Based on the total cap of expected support calculation method, we have the following lemma.

Lemma 1. *For a k-itemset X (where k=1) in DB, if $expSup^{Cap}(X) < minsup$, then any m-itemset Z (where $m > k$) in DB that contains X (i.e., $X \subset Z$) cannot be frequent.*

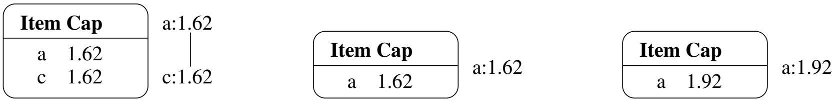
The above lemma provides us the opportunity to prune the CUF-tree further by removing any item if its total cap of expected support is less than the *minsup*. To save tree traversal cost in searching for such an item, we use the node traversal pointers to horizontally reach to that node in the CUF-tree and delete it. Hence, we can delete item b from the CUF-tree in Fig. 2(a) because $expSup^{Cap}(b) = 0.84 < minsup$. This results in a reduced tree as shown in Fig. 2(b). This tree pruning technique can save the mining time because we avoid mining those frequent items having infrequent extensions.

Let $F(t_j)$ be the set of frequent items in transaction t_j. Based on the tree construction mechanism described above, the following important properties hold in our CUF-tree:

1. The children list of a parent node in a CUF-tree contains one or more distinct items (i.e., there is no duplicate members in the list).
2. A CUF-tree registers the projection of $F(t_j)$ for t_j in DB only once.
3. The transaction cap in a node in a CUF-tree maintains the sum of transaction caps of all transactions that pass through or end at the node for all the nodes in the path from that node up to the root.
4. The total transaction cap of any node in a CUF-tree is greater than or equal to the sum of the total transaction cap values of its children.

Properties 1–3 are the most interesting properties of CUF-trees over UF-trees. As common prefixes of transactions are shared, the CUF-tree becomes more compact and it avoids repeated siblings of the same item. The following lemma states that a CUF-tree is a highly compact tree structure and its size can be similar to that of an FP-tree.

Lemma 2. *The size of a CUF-tree (without the root node) for a transaction database DB when using minsup is bounded by $\sum_{t_j \in DB} |F(t_j)|$.*



(a) for $\{e\}$-projected DB **(b)** for $\{c, e\}$-projected DB **(c)** for $\{c\}$-projected DB

Fig. 3. CUF-trees (built during the CUF-growth mining process)

Lemma 3. *Given a transaction DB and minsup, the complete set of all potential frequent itemsets can be obtained from a CUF-tree when applying minsup on DB.*

Based on Lemma 3, we mine our CUF-tree using a pattern-growth mining algorithm called **CUF-growth**.

3.2 CUF-growth: Mining Frequent Itemsets from the CUF-tree

The CUF-growth algorithm is designed in such a way that it deals with transaction caps (instead of occurrence frequencies as does in the FP-growth [8] algorithm). The basic operation of CUF-growth is constructing projected databases and recursively mining extensions of frequent itemsets. Once an itemset is found to be potentially frequent, CUF-growth generates the projected database based on the following property of a CUF-tree and lemma.

Property 1. To compute all potential frequent itemsets with suffix x, only the prefix sub-paths of nodes labelled x in the CUF-tree need to be accumulated. The nodes of every path should carry the transaction cap as that in the corresponding node x in the path.

Lemma 4. *Let X be a k-itemset (with $k \geq 1$) in DB, DB_X be an X-projected database, and Y be an itemset in DB_X. Then, $expSup^{Cap}(X \cup Y)$ in DB is equivalent to $expSup^{Cap}(Y)$ for the transactions in DB_X.*

Since the cap of expected support of an itemset X is an upper bound of the expected support of X and it satisfies the downward closure property, we apply CUF-growth to a CUF-tree to generate only the k-itemsets (with $k > 1$) whose caps of expected support are greater than or equal to the *minsup*. It will ensure that we do not miss any frequent itemset in the mining phase and the resulting set would be a superset of all frequent itemsets (i.e., the resulting set may contain some false positives, but *no* false negative will be in the resulting set). With an additional database scan, we can obtain the *exact* set of frequent itemsets from the resulting set. To get a better understanding of our CUF-growth algorithm, let us consider the following example, which uses the CUF-tree shown in Fig. 2(b).

Example 2. CUF-growth recursively mines the projected databases of all items in *I-list*. Hence, the $\{e\}$-projected database, as shown in Fig. 3(a) is constructed by accumulating the tree path $\langle a{:}1.62,\ c{:}1.62\rangle$. Note that the total cap for each node in the path is

Algorithm 2: Mining of "frequent" itemsets by CUF-growth

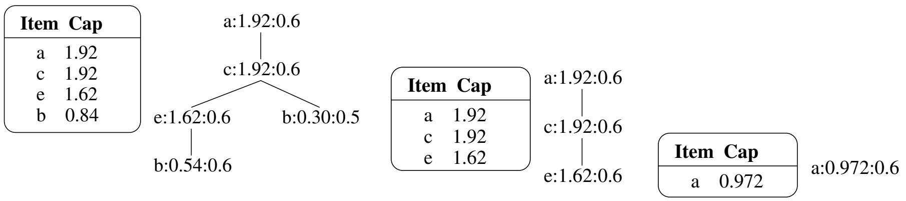
```
Input: A CUF-tree capturing items from $DB$ with $minsup$
Output: A set of "frequent" itemsets mined from the CUF-tree using $minsup$
begin
    for each item $\alpha \in$ I-list do
        $PB_\alpha$ = Build_PB(CUF-tree, $\alpha$), call Mine($PB_\alpha$, $\alpha$);

Function Build_PB(CUF-tree, $\alpha$)
begin
    for each node $N_\alpha$ of the last item of $\alpha$ in CUF-tree do
        project path $P_\alpha$ from the parent of $N_\alpha$ up to the root with $N_\alpha.t^{Cap}$ for each node
        in the conditional pattern-base $PB_\alpha$ of $\alpha$;
    let I-list$_\alpha$ be the I-list for $PB_\alpha$;
    return $PB_\alpha$;

Procedure Mine($PB_\alpha$, $\alpha$)
begin
    $CT_\alpha$ = Build_CT($PB_\alpha$);
    if $CT_\alpha \neq$ NULL then
        for each item $\beta \in$ I-list$_\alpha$ of $CT_\alpha$ do
            generate $\beta = \beta \cup \alpha$ as a candidate frequent itemset;
            $PB_\alpha$ = Build_PB($CT_\alpha$, $\beta$); call Mine($PB_\alpha$, $\beta$);

Function Build_CT($PB_\alpha$)
begin
    for each item $\beta \in$ I-list$_\alpha$ do
        if $expSup^{Cap}(\beta) < minsup$ then
            delete $\beta$ from I-list$_\alpha$; delete all $N_\beta$ nodes from $PB_\alpha$;
    return $CT_\alpha$ (which is the conditional tree constructed from $PB_\alpha$);
```

taken as the respective total transaction cap of e for the path. While projecting a path in $\{e\}$-projected *DB* from the original tree, the algorithm also calculates the cap of expected support for each item in the projected *DB*, as indicated in the *I-list* in the figure. Since the cap of expected supports of both items c and a are greater than the *minsup*, we first generate itemset $\{c, e\}$:1.62, and then proceed to construct $\{c, e\}$-projected database from the $\{e\}$-projected database of Fig. 3(a). The $\{c, e\}$-projected database is shown in Fig. 3(b). We generate itemset $\{a, c, e\}$:1.62 from this database, since item a's cap of expected support is found greater than the *minsup*. Because no further extension of itemset $\{a, c, e\}$ is possible, we return to the $\{c, e\}$-projected database and mine for the next item a (i.e., $\{a, e\}$-projected database) and generate itemset $\{a, e\}$:1.62. After finishing mining the $\{e\}$-projected database, we proceed to mine the $\{c\}$-projected database in similar fashion (as shown in Fig. 3(c)). We terminate the mining process when all items in the *I-list* are mined. The set of generated itemsets from the CUF-tree is $\{a\}$:2.3, $\{c\}$:2.2, $\{e\}$:1.8, $\{b\}$:1.4, $\{c, e\}$:1.62, $\{a, c, e\}$:1.62, $\{a, e\}$:1.62 and $\{a, c\}$:1.92. The CUF-growth mining algorithm is presented in Algorithm 2.

Among the itemsets generated in Example 2, $\{c, e\}$ and $\{a, c, e\}$ are false positives, since $expSup(\{c, e\})$=0.88 and $expSup(\{a, c, e\})$=0.432 are less than *minsup*. As indicated earlier in this section, with the third database scan, all false positives can be removed by calculating the actual expected support of each itemset in the resultant set.

4 Our CUF-growth* Algorithm: An Improvement to CUF-growth

In the previous section, we showed how CUF-tree provides an upper bound to expected support by capturing only the cap value (i.e., the two highest existential probability values for each node item). To tighten the upper bound, we present the **CUF-tree* structure** and the corresponding **CUF-growth* algorithm** in this section. A key difference between the CUF-tree and the CUF-tree* is that each node of the latter keeps (i) an item, (ii) the transaction cap, and (iii) the "bronze" value. Recall that, in CUF-trees, the upper bound is given by the transaction cap $P^{Cap}(t_j)$, which is the product of the two highest existential probability values. Here, the *"bronze" value* is the third highest existential probability value in a transaction. A *tighter upper bound* can then be given by the product of $P^{Cap}(t_j)$ and the "bronze" value. We have the following definition and theorem.



(a) Initial CUF-tree* **(b)** CUF-tree* for mining **(c)** for $\{c, e\}$-proj. DB

Fig. 4. CUF-trees* (built during the CUF-growth* mining process)

Definition 3. *The new upper bound of existential probability of any k-itemset X in t_j (denoted as $P(\widehat{X, t_j})$) is defined as follows. Let h be the transaction length, i.e., $h = |t_j|$. Let M_1, M_2, and M_3 be the three highest existential probability values in t_j. Then, $P(\widehat{X, t_j}) = P^{Cap}(t_j)$ if $h \leq 2$ (i.e., $P(\widehat{X, t_j}) = M_1$ if $h = 1$, $P(\widehat{X, t_j}) = M_1 \times M_2$ if $h = 2$), and $P(\widehat{X, t_j}) = P^{Cap}(t_j) \times$ ($\prod_{i=3}^{k} M_3$) if $h \geq 3$.*

Theorem 3. *Let M_3 denote the "bronze" value (the third highest existential probability value). The existential probability of any k-itemset X (with $k > 2$) in a transaction t_j is bounded above by the new upper bound: $P(X, t_j) \leq P^{Cap}(t_j) \times \prod_{i=3}^{k} M_3$, where X is a k-itemset with $k > 2$ and $X \subseteq t_j$.*

The CUF-tree* constructed for the dataset in Table 1 when using *minsup*=1 is shown in Figs. 4(a)–(b). If the prefix of a transaction t_j to be inserted in CUF-tree* is identical to an existing path, we store the maximum of the current bronze value of the node and the bronze value of t_j for each node in the path. Thus, each node in a CUF-tree* contains the highest M_3 value among all the transactions that pass through or end at that node, which eventually guarantees the tightness of the upper bound.

For the mining process of CUF-growth*, the first two levels (L_1 and L_2) are identical to the CUF-growth. From level 3 (L_k where $k \geq 3$), CUF-growth* multiplies the cap with the "bronze" value ($k-2$) times. As such, we obtain a tighter upper bound to expected support. The higher the level, the tighter the upper bound would be. Hence, fewer false positives are returned. The benefits of considering the bronze value can be observed in the example when CUF-growth* mining (Fig. 4(c)) is applied to the CUF-tree* of Fig. 4(b). While constructing the $\{c, e\}$-projected database (in Fig. 4(c)) from the $\{e\}$-projected database, the cap of the node (a:0.972:0.6) in $\{c, e\}$-projected database is calculated by multiplying 1.62 and 0.6 (respectively the cap and the bronze value of the node (c:1.62:0.6) in $\{e\}$-projected database). Note that, unlike CUF-growth, the CUF-growth* algorithm stops generating further patterns from the $\{c, e\}$-projected database because the cap of expected support of a in $\{c, e\}$-projected database is less than the *minsup*. Hence, itemset $\{a, c, e\}$ will not be generated. The set of generated itemsets from the CUF-tree* is $\{a\}$:2.3, $\{c\}$:2.2, $\{e\}$:1.8, $\{b\}$:1.4, $\{c, e\}$:1.62, $\{a, e\}$:1.62 and $\{a, c\}$:1.92. The above modification of CUF-growth* over CUF-growth algorithm can easily be adapted by including following conditional statement in between lines 6 and 7 of Algorithm 2 (i.e., in the for loop of line 6 and before the statement at line 7): "**if** α contains more than one item **then** $N_\alpha.t^{Cap} = N_\alpha.t^{Cap} \times N_\alpha.bronze\_value$;".

We show the performance of our proposed tree structure and mining algorithm in the next section.

5 Experimental Results

In this section, we present the experimental results on the proposed CUF-growth, its variant CUF-growth*, and existing algorithms (e.g., UF-growth [13], UFP-growth [1] and UH-mine [1]). Experiments were conducted on real and synthetic datasets. The synthetic datasets, which are generally sparse, are generated within a domain of 1000 items by the dataset generator developed by IBM Almaden Research Center [2]. We assigned an existential probability from the range (0,1] to each item in each transaction in each dataset. The name of a dataset indicates its characteristics information. For example, the dataset named u100k10L_10_100 contains 100K transactions with average transaction length 10, and each item in a transaction is associated with an existential probability value within a range from 10% to 100%. We also considered several real dense datasets such as *mushroom* and *connect4*. However, because of space constraint, we only present here the results on *mushroom* and some IBM datasets.

All programs were written in C and run with UNIX on a quad-core processor with 1.3GHz. Unless otherwise specified, runtime includes CPU and I/Os for*I-list* construction, tree construction, mining, and false positive calculation (for UFP-growth, CUF-growth and CUF-growth*). The results shown in this section are based on the average of multiple runs for each case. In all of the experiments, the CUF-trees were constructed in descending order of expected support of items.

5.1 Compactness of the CUF-tree

In the first experiment, we evaluated the compactness of our CUF-tree in terms of the number of nodes. The UF-tree, UFP-tree, CUF-tree and CUF-tree* all arrange items in descending order of expected supports. Hence, the tree structure and the size of UFP-trees and CUF-trees were similar. However, as a UFP-tree stores the extra cluster information, the size of a UFP-tree node is usually larger than that of a CUF-tree node. The size of a UF-tree was larger than the other three because the former may contain multiple nodes of the same item (if having different existential probability values) under a common parent.

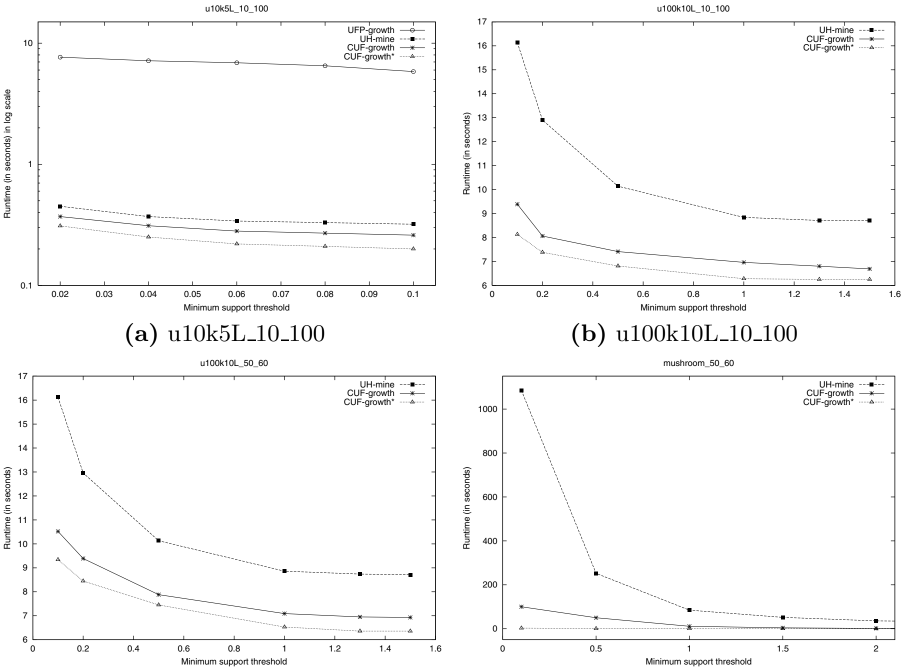
Table 2. Compactness of CUF-trees (#tree nodes)

Dataset	*minsup*	**UF-tree**	**CUF-tree**	*minsup*	**UF-tree**	**CUF-tree**
u10k5L_10_100	0.02	52768	6943 (13.17%)	1.0	50673	6877 (13.57%)
u100k10L_50_60	0.1	810735	90097 (11.11%)	1.5	793943	89971 (11.33%)
u100k10L_10_100	0.1	884240	90097 (10.18%)	1.5	867169	89969 (10.37%)
mushroom_50_60	0.1	121205	8108 (6.68%)	5	108289	5008 (4.62%)
mushroom_10_100	0.5	160381	2726 (1.7%)	10	135591	2046 (1.5%)

The results of the experiments on different datasets over several respective *minsup* values (in percentage) are presented in Table 2. The fourth and last columns of the table represent the CUF-tree nodes in percentage of UF-tree nodes. To observe the compactness of our CUF-tree (irrespective of support threshold and dataset characteristics), for each dataset, we showed the sizes of both trees on a low and a high *minsup* values. Since both CUF-tree and CUF-tree* contain exactly the same number of nodes for a given dataset and *minsup*, we only show the node count for CUF-tree in the table. Notice that it is common in both UF-tree and CUF-tree that, for a specific dataset, they require respectively similar number of nodes with variations of *minsup*. However, as expected, our CUF-tree is more compact than the UF-tree in both sparse and dense cases. The memory gain of CUF-trees over UF-tree is much promising in dense datasets. For example, the size of the CUF-tree was only 10.18% of that of the UF-tree for *minsup*=0.1 for u100k10L10_100, while the gain increases significantly as of 1.5% on mushroom10_100 for *minsup*=10 for the dense dataset *mushroom*. The reason is that, for dense datasets, the CUF-tree is more likely to have more chance of sharing paths for common prefixes, irrespective of different existential probabilities of each item in different transactions. However, the UF-tree did not achieve such tree compactness and follows different paths for each combination of existential probabilities of the same items.

5.2 Runtime

We compared the runtime of our CUF-growth with those of UH-mine and UFP-growth. Fig. 5 shows that CUF-growth* outperformed CUF-growth.



(a) u10k5L_10_100 **(b)** u100k10L_10_100

(c) u100k10L_50_60 **(d)** mushroom_50_60

Fig. 5. Comparison on execution times

The reason is due to the use of *bronze* value, which leads to fewer number of false positives (which is verified in the next experiment) and eventually minimizes the overall runtime. When mining, although CUF-growth* incurred an additional computation cost for calculating the cap of expected supports for k-itemsets (with $k > 2$), it was relatively insignificant compared to the gain brought by the use of the *bronze* value.

Fig. 5(a) shows the results for u10k5L_10_100, which is a small dataset (in terms of number of transaction and transaction length) compared to other datasets we used. Note that, in this dataset, CUF-growth and CUF-growth* significantly outperformed UFP-growth (note the y-axis of the graph is in log scale). Even though the UH-mine also performed considerably better than UFP-growth for the dataset, it was still worse than CUF-growth. While considering larger datasets, UFP-growth took high runtimes to plot. Moreover, Aggarwal et al. [1] showed that UH-mine outperforms UFP-growth in execution time. Hence, for the other figures (i.e., Figs. 5(b)–(d)), we avoid plotting the extreme high readings of UFP-tree. We compare our CUF-growth and CUF-growth* with UH-mine.

The performance gain of CUF-growth over UH-mine on large datasets was impressive. The reason of such gain is that, even though UH-mine finds the exact set of frequent itemsets when mining the extension x of a prefix part X, it suffers from the high computation cost for calculating the expected support of X on-the-fly in the transactions where it appears. Such computation may become more costly when mining for large number of itemsets and long itemsets. For example, the gap between the performance curves of CUF-growth and UH-mine becomes comparatively wide for low *minsup* (Figs. 5(b)–(c)) and dense dataset (Fig. 5(d)).

5.3 Number of False Positives

It is evidential that CUF-trees, CUF-trees*, and UFP-trees are usually compact because they arrange tree nodes in descending order of expected supports. However, all of them generate some false positives. Their overall performances depend on the number of generated false positives. In this experiment we investigated the number of false positives generated by each of above trees. We obtained similar results over different datasets in false positive computation. Hence, because of space limitation, in Table 3 we present only the results over two datasets (i.e., u10k5L_10_100 and u100k10L_50_60) for one *minsup* value. The last three columns of the table indicate the number of false positives in the percentage of total itemsets generated by respective algorithms (e.g., 62.23% of the total number of itemsets generated by UFP-tree are false positives on u10k5L_10_100 for *minsup*=0.1).

Table 3. Comparison on False Positives (in terms of % of total #itemses)

Dataset	*minsup*	**UFP-growth**	**CUF-growth**	**CUF-growth***
u10k5L_10_100	0.1	62.23%	28.08%	26.40%
u100k10L_50_60	1.0	71.91%	17.56%	16.86%

The table also indicates that CUF-growth* generates fewer false positives when compared to CUF-growth. This is because during mining k-itemsets ($k > 2$) by applying the *bronze* value in cap computation, the CUF-growth* constantly tightens the upper bound of the expected support of the itemset. However, it can be observed from the table that both CUF-growth and CUF-growth* remarkably reduced the number of the false positives generated. The primary reason of this improvement is that the UFP-growth uses the cluster summary information stored at each node to calculate the false positives, which fails to guarantee to obtain a much tighter upper bound for expected support calculation. In a UFP-tree, if a parent has several children, then each child will use the higher cluster values in the parent to generate the total expected support. If the total number of existential probability value of that child is still lower than that of the parent's higher cluster value, the expected support of the path with this parent and a child will be high, which results in more false positives in the long run.

The above experimental results show that our CUF-growth outperformed the state-of-the-art algorithms in generating frequent itemsets from uncertain data irrespective of low (e.g., 0.5 to 0.6) or high (e.g., 0.1 to 1.0) distribution of existential probabilities.

Due to space limitation, we omit the experimental results that show the scalability of our proposed algorithm. As an ongoing work, we plan to conduct more experiments (e.g., using additional datasets like gazelle sparse dataset).

6 Conclusions

Over the past few years, several algorithms have been proposed to mine frequent itemsets from uncertain data. The UF-growth algorithm may build a large UF-tree because nodes are shared only if their ⟨item, existential probability value⟩ are identical. To reduce the number of tree nodes, UFP-growth clusters nodes with the same item but similar existential probability values. However, unlike UF-growth (which is an exact algorithm), UFP-growth is an approximate algorithm (which may generate false positives). To further reduce the number of tree nodes (when compared with the UF-tree) and false positives (when compared with UFP-growth), we proposed CUF-growth and CUF-growth* in this paper. By capturing the cap (i.e., product of the two highest existential probability values in a transaction) in the CUF-tree, our CUF-growth algorithm gives a tight upper bound on the expected support of itemsets. Hence, it reduces the number of false positives. It efficiently finds frequent itemsets while keeping the number of nodes in the CUF-tree be the same as that in the FP-tree. Our CUF-growth* algorithm further reduces the number of false positives by keeping the third highest existential probability of items in a transaction. This leads to a tighter upper bound on the expected support of itemsets when mining frequent itemsets from uncertain data.

Acknowledgement. This project is partially supported by NSERC (Canada) and University of Manitoba.

References

1. Aggarwal, C.C., Li, Y., Wang, J., Wang, J.: Frequent pattern mining with uncertain data. In: ACM KDD 2009, pp. 29–37 (2009)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: VLDB 1994, pp. 487–499 (1994)
3. Bernecker, T., Kriegel, H.-P., Renz, M., Verhein, F., Zuefle, A.: Probabilistic frequent itemset mining in uncertain databases. In: ACM KDD 2009, pp. 119–127 (2009)
4. Calders, T., Garboni, C., Goethals, B.: Approximation of frequentness probability of itemsets in uncertain data. In: IEEE ICDM 2010, pp. 749–754 (2010)
5. Calders, T., Garboni, C., Goethals, B.: Efficient Pattern Mining of Uncertain Data with Sampling. In: Zaki, M.J., Yu, J.X., Ravindran, B., Pudi, V. (eds.) PAKDD 2010, Part I. LNCS (LNAI), vol. 6118, pp. 480–487. Springer, Heidelberg (2010)
6. Chui, C.-K., Kao, B., Hung, E.: Mining Frequent Itemsets from Uncertain Data. In: Zhou, Z.-H., Li, H., Yang, Q. (eds.) PAKDD 2007. LNCS (LNAI), vol. 4426, pp. 47–58. Springer, Heidelberg (2007)

7. Eavis, T., Zheng, X.: Multi-Level Frequent Pattern Mining. In: Zhou, X., Yokota, H., Deng, K., Liu, Q. (eds.) DASFAA 2009. LNCS, vol. 5463, pp. 369–383. Springer, Heidelberg (2009)
8. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: ACM SIGMOD 2000, pp. 1–12 (2000)
9. Kiran, R.U., Reddy, P.K.: An Alternative Interestingness Measure for Mining Periodic-Frequent Patterns. In: Yu, J.X., Kim, M.H., Unland, R. (eds.) DASFAA 2011, Part I. LNCS, vol. 6587, pp. 183–192. Springer, Heidelberg (2011)
10. Lakshmanan, L.V.S., Leung, C.K.-S., Ng, R.T.: Efficient dynamic mining of constrained frequent sets. ACM TODS 28(4), 337–389 (2003)
11. Leung, C.K.-S., Hao, B.: Mining of frequent itemsets from streams of uncertain data. In: IEEE ICDE 2009, pp. 1663–1670 (2009)
12. Leung, C.K.-S., Jiang, F.: Frequent Pattern Mining from Time-Fading Streams of Uncertain Data. In: Cuzzocrea, A., Dayal, U. (eds.) DaWaK 2011. LNCS, vol. 6862, pp. 252–264. Springer, Heidelberg (2011)
13. Leung, C.K.-S., Mateo, M.A.F., Brajczuk, D.A.: A Tree-Based approach for Frequent Pattern Mining from Uncertain Data. In: Washio, T., Suzuki, E., Ting, K.M., Inokuchi, A. (eds.) PAKDD 2008. LNCS (LNAI), vol. 5012, pp. 653–661. Springer, Heidelberg (2008)
14. Lin, C.-W., Hong, T.-P., Lu, W.-H.: A new tree structure for mining frequent itemsets from uncertain databases. In: Nat'l Conf. on Fuzzy Theory and its App., pp. 575–579 (2009)
15. Pei, J., Han, J., Lu, H., Nishio, S., Tang, S., Yang, D.: H-Mine: hyper-structure mining of frequent patterns in large databases. In: IEEE ICDM 2001, pp. 441–448 (2001)
16. Zhang, Q., Li, F., Yi, K.: Finding frequent items in probabilistic data. In: ACM SIGMOD 2008, pp. 819–832 (2008)

On the Decidability and Complexity of Identity Knowledge Representation

Klaus-Dieter Schewe[1] and Qing Wang[2]

[1] Software Competence Center Hagenberg and Johannes-Kepler-University Linz, Austria
kd.schewe@scch.at, kd.schewe@faw.at

[2] Department of Information Science, University of Otago, Dunedin, New Zealand
qing.wang@otago.ac.nz

Abstract. Identity knowledge is the knowledge that relates to various aspects of the identification of real-world objects. It can be acquired through the process of identifying objects from a knowledge management point of view. In this paper we present a simple yet expressive framework for representing identity knowledge. Knowledge patterns, as the building blocks of the framework, have the capability of capturing identity knowledge at an arbitrary level of abstraction. However, the combined use of pattern formula and pattern relation in knowledge patterns may yield disjunction and a restricted form of negation. We thus investigate the containment problem of knowledge patterns to find a decision procedure for containment and equivalence between knowledge patterns. Our result shows that the containment problem for knowledge patterns is not only decidable but also tractable.

1 Introduction

The identification problem pervasively exists in many research areas such as scientific communities (e.g., [1,20]), information integration (e.g., [10,14]), data cleaning (e.g., [7]), service-oriented architecture (e.g., [2]), etc. It is to determine which objects in a system correspond to the same object in the real-world. When restricted by the availability of sufficient identity information, determining whether or not two representations correspond to the same real-world object is not always possible. However, it would be desired to find an approach that can provide approximate identity and meanwhile allow us to improve the accuracy of identity over time. Identity knowledge management turns out to a promising direction to achieve this. Whenever two objects with varied representations are identified to be the same real-world object, capturing the knowledge of why they are or are not the same object is at least equally important to simply unifying them into one identity. Take the records in Table 1 for example, and assume that we acquire the following knowledge from certain source:

(a) Sue Lee at the Bioethics Centre of the University of Otago has studied at the Department of Philosophy of the University of Otago;

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 288–302, 2012.

© Springer-Verlag Berlin Heidelberg 2012

(b) Sue Lee at the Department of Philosophy of the University of Otago has not previously worked or studied at Massey University;

(c) Sue Lee at the Bioethics Centre of the University of Otago changed her surname to Maneth after marriage.

If the above identity knowledge statements and the like can be represented by a knowledge model built upon application data, we would be empowered to understand how object identities are linked and to share identity knowledge across multiple communities. Furthermore, identity knowledge can offer many advantages for solving the identification problem in a more effective and collaborative way.

Table 1. Sample records in the relation PERSON

ID	Name	Department	University
i_1	Sue Lee	Department of Philosophy	University of Otago
i_2	Sue Maneth	Bioethics Centre	University of Otago
i_3	Sue Lee	School of History and Philosophy	Massey University
i_4	Sue Lee	Bioethics Centre	University of Otago

In this paper our first contribution is to introduce a simple yet expressive framework for representing identity knowledge. A knowledge model is comprised of a number of knowledge patterns that work accumulatively to deduce object identities. Each knowledge pattern has the ability of expressing identity knowledge at various levels of abstractions, ranging from a generic perspective expressed as the pattern formula to more specific perspectives expressed as instantiations of the pattern formula. Moreover, identity knowledge at any level of abstraction can be specified either as a default rule to deduce object identities or as an exception to default rules in the same knowledge pattern to exclude certain object identities from consideration. In order to enhance the expressiveness of the framework, we permit that different knowledge patterns may associate with the equivalent pattern formulae. This is due to the fact that a knowledge pattern cannot always capture the same identity knowledge represented by multiple knowledge patterns that have the equivalent pattern formulae. In doing so, we obtain a framework that has sufficient expressive power to capture application-specific knowledge from various aspects of the identification problem.

However, the gain in expressive power of the framework does not come for free. As each knowledge pattern associates with not only a pattern formula but also a pattern relation that accommodates application-specific identity knowledge, the containment problem for knowledge patterns naturally arises – is there an efficient way to find that one knowledge pattern has captured all identity knowledge represented by another knowledge pattern?

Our second contribution is the decidability and complexity result of the containment problem for knowledge patterns. We show that the containment problem for knowledge patterns is not only decidable but also tractable, even though the logical formalisation of a knowledge pattern involves conjunction, disjunction and also a restricted form of negation. Based on the Homomorphism Theorem for containment of conjunctive queries, we provide a characterisation for containment of knowledge patterns. It leads to the results that the containment problem for knowledge patterns is NP-complete in terms of the size of the knowledge patterns, but it is in PTIME in terms of data complexity. In a sense similar to the query containment problem that has important applications in query optimization, the result of the containment problem for knowledge patterns would allow us to efficiently discover redundancy among knowledge patterns and thus helps to optimise identity knowledge management.

The remainder of the paper is structured as follows. We present the related work in the areas of object identification and query containment in Section 2. Section 3 introduces the formal framework we propose for representing identity knowledge. In Section 4 we discuss how to minimise a knowledge pattern from two aspects: pattern formula and pattern relation. Then, we investigate the containment problem for knowledge patterns in Section 5. Section 6 analyses the complexity of the containment problem for knowledge patterns. We briefly conclude the paper in Section 7.

2 Related Work

Over the past decades, there has been considerable research efforts made on solving the identification problem (e.g., [7,9,14,19,22,24,25]). The most dominant research approaches are the use of similarity functions, in which approximate string matching algorithms and adaptive algorithms with active learning are often implemented (e.g., [22,25]), and the identification problem was mainly approached from a probabilistic point of view. Recently, [6,7] proposed a transformation-based framework combining context-free grammar rules with database querying to manipulate representational variations of objects, in which common domain knowledge for varied representation can be incorporated. Nevertheless, little work has been done on the identification problem to capture identity knowledge at various levels of abstraction, which indeed becomes increasingly important for accurately identifying objects by knowledge sharing or working collaboratively within a wide range of communities.

Our investigation on the containment problem for knowledge patterns extends the previous results on the query containment problem, which is one of the cental issues in database theory and has important impact on many areas such as query optimization and knowledge verification [3,4,11,17,18,21]. It is well-known that query containment is decidable for conjunctive queries [12,4] and union of conjunctive queries [21], but not decidable for first-order queries and Datalog queries [11,23]. The papers (e.g.,[15,16,26,28]) studied the containment problem for conjunctive queries with inequality. The complexity of the containment problem for conjunctive queries is NP-complete, which is measured in terms of the size of the query rather than the data. Since queries are much smaller than data in general, and the containment problem for conjunctive queries is shown to be exactly the same as the query evaluation problem that can be computed in polynomial time in terms of data complexity [3], the containment problem of conjunctive queries is tractable. Nevertheless, the complexity of the containment problem for conjunctive queries with inequality is Π_2^P-complete [26]. Our work on the containment problem for knowledge patterns identifies a new class of queries with conjunction, union and a restricted form of negation, whose containment problem can polynomially reduce to the containment problem of conjunctive queries. Thus, deciding containment between these queries is efficient.

3 Formal Framework

Let us fix a family $\{D_i\}_{i \in I}$ of pairwise disjoint basic domains, an identity domain $D_O \in \{D_i\}_{i \in I}$, a database schema $\mathcal{S}$ over $\{D_i\}_{i \in I}$ consisting of a set of relation names, and an equivalence relation $\textsc{Iden} \notin \mathcal{S}$ over D_O.

Definition 1. *A* knowledge pattern *P is a pair $\langle \varphi, r \rangle$ consisting of*

- *a* pattern formula *φ that has the form $\textsc{Iden}(x, y) \leftarrow \psi(x_1, \ldots, x_n, x, y)$, in which ψ is a conjunction of atoms, and x and y are variables over D_O, and*
- *a* pattern relation *r of the arity $n+1$ with n attributes $A_1, \ldots, A_n$ that are in 1-1 correspondence to the variables $x_1, \ldots, x_n$ in φ, expressed as $\iota(x_i) = A_i$ $(i = 1, \ldots, n)$, plus an attribute A^* with domain $\{+, -\}$.*

To support the flexibility of abstraction levels, each knowledge pattern uses a pattern formula and a pattern relation in a combined way such that variables of the pattern formula are bound to attributes of the pattern relation. The formula φ is generic (i.e., containing no constants) in the sense of the genericity principle defined for database queries [5]. The relation r may contain constants from the base domains of its attributes and the symbol λ. We use the symbol λ to indicate that any appropriate value may occur in its place. Two possible dimensions (i.e., including as "+" and excluding as "–") can be specified via the attribute A^* of each pattern relation.

Given a knowledge pattern $P = \langle \varphi, r \rangle$, where φ is in the form of $\textsc{Iden}(x, y) \leftarrow \psi(x_1, \ldots, x_n, x, y)$, a set of specific queries are obtained by substituting the occurrences of bounded variables in $\psi(x_1, \ldots, x_n, x, y)$ with their associated attribute values in r. Let $t.A$ denote the value of the attribute A in a tuple t of the relation r. Then each tuple t in the relation r "generates" exactly one specific query from $\psi(x_1, \ldots, x_n, x, y)$ by,

- for each variable x_k with $t.\iota(x_k) \neq \lambda$, replacing x_k in the predicates of ψ with $t.\iota(x_k)$, and
- for each variable x_k with $t.\iota(x_k) = \lambda$, adding x_k after an existential quantification that is before ψ.

We distinguish two kinds of specific queries according to the value of attribute A^*. A specific query is an *in-query* if $t.A^* = +$ for the tuple t that generates it; otherwise it is an *ex-query*.

The following example illustrates how identity knowledge can be captured by using knowledge patterns.

Example 1. Consider the relation Person in Table 1 and suppose that we have the relation Iden = {ID$_1$,ID$_2$} for storing equivalent identities of persons.

- The following pattern $P_1 = \langle \varphi_1, r_1 \rangle$ describes that two persons are identical if they have the same name and are affiliated with certain departments, as stipulated in r_1. It captures the knowledge statements (a) and (b) we mentioned in Section 1.

$$\varphi_1 : \textsc{Iden}(x, y) \leftarrow \textsc{Person}(x, z_1, x_2, x_3) \wedge \textsc{Person}(y, z_1, y_2, y_3)$$

r_1 :

A_{z_1}	A_{x_2}	A_{x_3}	A_{y_2}	A_{y_3}	A^*
Sue Lee	Bioethics Centre	University of Otago	Department of Philosophy	University of Otago	+
Sue Lee	λ	Massey University	Department of Philosophy	University of Otago	−

(*continued*)

- The following pattern $P_2 = \langle \varphi_2, r_2 \rangle$ describes that two persons are identical if they are affiliated with the same department and have certain name variations, as stipulated in r_2. It captures the knowledge statement (c) discussed in Section 1, i.e., Sue Lee and Sue Maneth at the Bioethics Centre of the University of Otago are the same person.

$$\varphi_2 : \textsc{Iden}(x, y) \leftarrow \textsc{Person}(x, x_1, z_2, z_3) \wedge \textsc{Person}(y, y_1, z_2, z_3)$$

r_2 :

A_{z_2}	A_{z_3}	A_{x_1}	A_{y_1}	A^*
Bioethics Centre	University of Otago	Sue Lee	Sue Maneth	+

The knowledge pattern $P_1 = \langle \varphi_1, r_1 \rangle$ has the following specific queries:

- $q_1^+(x, y) \equiv$
  $\textsc{Person}(x, \text{“Sue Lee”}, \text{“Bioethics Centre”}, \text{“University of Otago”}) \wedge$
  $\textsc{Person}(y, \text{“Sue Lee”}, \text{“Department of Philosophy”}, \text{“University of Otago”})$
- $q_1^-(x, y) \equiv$
  $\exists x_2.\textsc{Person}(x, \text{“Sue Lee”}, x_2, \text{“Massey University”}) \wedge$
  $\textsc{Person}(y, \text{“Sue Lee”}, \text{“Department of Philosophy”}, \text{“University of Otago”})$

where $q_1^+(x, y)$ is an in-query and $q_1^-(x, y)$ is an ex-query generated by the first and second tuples of r_1, respectively.

Let Σ_P^+ and Σ_P^- be the set of in-queries and the set of ex-queries of the knowledge pattern P, respectively, and $\Sigma_P = \Sigma_P^+ \cup \Sigma_P^-$. Then a *(knowledge) rule* of P, denoted as R^P, has an expression of the form

$$\textsc{Iden}(x, y) \leftarrow \bigvee_{q^+(x,y) \in \Sigma_P^+} q^+(x, y) \wedge \neg \bigvee_{q^-(x,y) \in \Sigma_P^-} q^-(x, y).$$

We use $rhs(R)$ to denote the right hand side of R. Let ν be a valuation over variables of R, i.e., a total function from variables $\{x, y\}$ to constants in $\bigcup\{D_i\}_{i \in I}$. Then the *interpretation* of rule R^P over a database instance I, denoted as $R^P(I)$, is defined as follows:

$$R^P(I) = \{\textsc{Iden}(\nu(x), \nu(y)) \mid rhs(R^P) \text{ is evaluated to true in } I \text{ under } \nu\}.$$

A *knowledge model* M is associated with a finite, non-empty set of knowledge patterns. The *program* of M consists of knowledge rules that have one-to-one correspondence with knowledge patterns of M. Semantically, the program of a knowledge model is interpreted in the same way as a program in Datalog with negation under the inflationary semantics [3]. Let Λ be the program of a knowledge model $M = \{P_1, ..., P_k\}$ and I be a database instance of $\mathcal{S}$. Then we have μ as an *inflationary fixpoint operator* such that $\mu(\Lambda)$ over I defines the identity relation that is the limit of the sequence $\{J_n(\textsc{Iden})\}_{n \geq 0}$ defined by

$$\begin{aligned} J_0(\textsc{Iden}) &= \emptyset \\ &\ldots \\ J_n(\textsc{Iden}) &= J_{n-1}(\textsc{Iden}) \cup \bigcup_{1 \leq i \leq k} R^{P_i}(J_{n-1}), \end{aligned}$$

where $R^{P_i}(J_{n-1})$ denotes the result of evaluating R^{P_i} on the instance J_{n-1} over $\mathcal{S} \cup \{\textsc{Iden}\}$ whose restriction to $\mathcal{S}$ is I and restriction to $\{\textsc{Iden}\}$ is $J_{n-1}(\textsc{Iden})$. Tuples in the relation Iden are created in a cumulative rather than destructive way, until a fixpoint of Iden is reached. The inflationary semantics of the program of a knowledge model guarantees termination of the computation in time polynomial in the size of the database [3]. We restrict Iden to be the only intensional predicate symbol appearing in the left hand side of a knowledge rule. Nevertheless, Iden can still appear in the right hand side. In doing so, it allows us to incorporate recursion into the process of deducing identity knowledge.

Example 2. Suppose that we have Publication = {ID, Title} and Author = {PID, PubID, Order} where PID of Author references ID of Person as shown in Table 1, and PubID of Author references ID of Publication. The following knowledge pattern P_3 has Iden in the both sides of its knowledge rules.

- The pattern $P_3 = \langle \varphi_3, r_3 \rangle$ describes that two persons are identical if they have the same name and both co-authored with another author.

$$\begin{aligned} \varphi_3 : \textsc{Iden}(x, y) \leftarrow\ & \textsc{Person}(x, z_1, x_2, x_3) \wedge \textsc{Author}(x, z_2, x_4) \wedge \\ & \textsc{Person}(y, z_1, y_2, y_3) \wedge \textsc{Author}(y, z_3, y_4) \wedge \\ & \textsc{Author}(z', z_2, x_5) \wedge \textsc{Author}(z, z_3, y_5) \wedge \textsc{Iden}(z, z') \end{aligned}$$

r_3 :

A_{z_1}	A_{z_2}	A_{z_3}	A_z	$A_{z'}$	A_{x_2}	A_{x_3}	A_{x_4}	A_{x_5}	A_{y_2}	A_{y_3}	A_{y_4}	A_{y_5}	A^*
λ	λ	λ	λ	λ	λ	λ	λ	λ	λ	λ	λ	λ	$+$

In order to cope with reflexivity, symmetry and transitivity of the equivalence relation Iden, every knowledge model must contain knowledge patterns that reflect these properties.

4 Minimising Knowledge Pattern

Minimising knowledge patterns of a knowledge model is important for improving the efficiency of evaluating knowledge patterns, and consequently, the overall performance of a knowledge model can be improved. Since a knowledge pattern is constituted by a pattern formula and a pattern relation, the minimisation of a knowledge pattern has two aspects: (1) finding the minimisation of the pattern formula, and (2) removing redundant tuples in the pattern relation.

Given two formulae φ_1 and φ_2 over the same schema $\mathcal{S}$, φ_1 is *contained* in φ_2 (denoted as $\varphi_1 \subseteq \varphi_2$) if, for each database instance I of $\mathcal{S}$, $\varphi_1(I) \subseteq \varphi_2(I)$, where $\varphi(I)$ denotes the interpretation of φ in I. φ_1 and φ_2 are *equivalent* (denoted as $\varphi_1 \equiv \varphi_2$) if $\varphi_1 \subseteq \varphi_2$ and $\varphi_2 \subseteq \varphi_1$. Conjunctive formulae, as discussed in [3], enjoy several interesting properties, e.g., determining equivalence and containment between conjunctive formulae is NP-complete and query minimisation is NP-hard. Since pattern formulae are indeed conjunctive formulae, finding the minimisation of a pattern formula can be handled in the same way of tableau query minimization and we thus omit further discussion on this aspect.

For the second aspect, i.e., redundant tuples in a pattern relation, let us start with the following example.

Example 3. Consider a pattern $P_4 = \langle \varphi_4, r_4 \rangle$ where φ_4 is the same as φ_1 defined in Example 1 and r_4 is presented as below.

r_4 :

A_{z_1}	A_{x_2}	A_{x_3}	A_{y_2}	A_{y_3}	A^*	
λ	λ	λ	λ	λ	+	t_1
λ	λ	Massey University	λ	λ	+	t_2
Sue Lee	λ	University of Auckland	λ	λ	+	t_3
Sue Lee	λ	λ	λ	λ	−	t_4
Sue Lee	λ	Massey University	λ	λ	−	t_5

Then we can see that the in-query generated by the tuple t_1 contains the in-queries generated by the tuples t_2 and t_3, and the ex-query generated by the tuple t_4 contains the ex-query generated by the tuple t_5. Hence, the tuples t_2, t_3 and t_5 are redundant in the pattern P_4. In other words, if removing the tuples t_2, t_3 and t_5 from the relation r_4, we can obtain the relation $r_4^{'}$ as below, and the identity knowledge captured by the pattern $\langle \varphi_4, r_4^{'} \rangle$ remains the same as captured by the pattern $\langle \varphi_4, r_4 \rangle$.

$r_4^{'}$:

A_{z_1}	A_{x_2}	A_{x_3}	A_{y_2}	A_{y_3}	A^*
λ	λ	λ	λ	λ	+
Sue Lee	λ	λ	λ	λ	−

To formalise redundant tuples in a pattern relation, we introduce the notion of subsumption between tuples under the assumption that constants and λ are partially ordered, i.e., $a \preceq \lambda$ for any constant a. We use $attr(r)$ to denote the set of attributes of a relation r. Let t_1 and t_2 be tuples that have the same set $attr(r)$ of attributes. Then we say that t_1 *subsumes* t_2 (denoted as $t_2 \sqsubseteq t_1$) if $t_2.A \preceq t_1.A$ holds for each attribute $A \in attr(r)$.

Definition 2. *Let t_1 and t_2 be two tuples of a pattern relation r. Then we have,*

- t_1 upward-subsumes t_2 *(denoted as $t_2 \Subset_{\uparrow} t_1$) if, for all attributes $A \in attr(r)$,*

$$\begin{cases} t_2.A = + \textit{ and } t_1.A = - & \textit{if} \quad A = A^*; \\ t_2.A \preceq t_1.A & \textit{otherwise.} \end{cases}$$

A pattern relation without redundant tuples should satisfy the following minimality property.

Definition 3. *Let r be a pattern relation. Then r satisfies the* minimality property *iff it satisfies the condition* $\bigwedge_{t_i \neq t_j \wedge t_i \in r \wedge t_j \in r} t_i \not\sqsubseteq t_j \wedge t_i \not\Subset_{\uparrow} t_j$.

A pattern relation r is *well-defined* if r is minimal and contains at least one tuple t with $t.A^* = +$. If the pattern relation of a knowledge pattern is not well-defined, then the knowledge pattern should be automatically removed from the knowledge model. This is because we require that every knowledge pattern can only yield positive object identities (i.e., two objects refer to the same real-world object) in its final result, rather than negative object identities (i.e., two objects do not refer to the same real-world object). Furthermore, by verifying the minimality property on a pattern relation, we are able to detect and eliminate redundant tuples, which can organise diverse identity knowledge acquired in different times, by different people and under different reasons more effectively.

Example 4. The relation r_4 in Example 3 does not satisfy the minimality property because we have $t_1 \sqsubseteq t_2$, $t_1 \sqsubseteq t_3$, $t_4 \sqsubseteq t_5$ and $t_3 \Subset_{\uparrow} t_4$. However, the relation $r_4^{'}$ satisfies the minimality property.

The following lemma can be easily proven in accordance with the definition of minimality property.

Lemma 1. *Let $P = \langle \varphi, r \rangle$ be a knowledge pattern and r be minimal. Then*

- *$\varphi_1 \nsubseteq \varphi_2$ holds for any two different φ_1 and φ_2 from Σ_P^+, and*
- *$\varphi_1 \nsubseteq \varphi_2$ holds for any $\varphi_1 \in \Sigma_P^+$ and $\varphi_2 \in \Sigma_P^-$.*

In the sequel knowledge patterns are considered to have the minimal pattern relations, unless otherwise stated.

5 The Containment Problem

Now we study the containment problem for knowledge patterns. Given two knowledge patterns P_1 and P_2 over the same schema $\mathcal{S}$, P_1 is *contained* in P_2, denoted as $P_1 \subseteq P_2$, if, for each database instance I of $\mathcal{S}$, $R^{P_1}(I) \subseteq R^{P_2}(I)$ holds, and similarly P_1 and P_2 are *equivalent*, denoted as $P_1 \equiv P_2$, if $P_1 \subseteq P_2$ and $P_2 \subseteq P_1$ both hold. The *containment problem* for knowledge patterns is to determine whether or not $P_1 \subseteq P_2$ holds for all instances of S.

We will show that, although knowledge patterns involve the union of conjunctive queries and also a restricted form of negation, the containment problem for knowledge patterns is not only decidable but also tractable. We use $sym(\varphi)$ to denote the set of all variables and constants occurring in a formula φ. Let φ_1 and φ_2 be formulae over the same schema. A *homomorphism* from φ_1 to φ_2 is a function $\theta : sym(\varphi_1) \mapsto sym(\varphi_2)$ such that: (1) $\theta(a) = a$ for every constant $a \in sym(\varphi_1)$, (2) $\theta(x) = y$ for every variable $x \in sym(\varphi_1)$, and (3) for every predicate $p(x_1, \dots, x_n)$ of φ_1, $\theta(p(x_1, \dots, x_n)) = p(\theta(x_1), \dots, \theta(x_n))$ is a predicate of φ_2. The Homomorphism Theorem [3] provides a characterization for containment of conjunctive queries.

Theorem 1. (Homomorphism Theorem [3]) *Let φ and ϕ be conjunctive queries over the same schema. Then $\varphi \subseteq \phi$ iff there exists a homomorphism from ϕ to φ.*

In the following we present a characterisation for containment of knowledge patterns. Our theorem is built upon Theorem 1 (i.e. the Homomorphism Theorem for conjunctive queries), Lemma 1 and the following two lemmata.

Lemma 2. *Let $\Sigma_P = \{\varphi_1, \dots, \varphi_n\}$ be a finite set of queries associated with a knowledge pattern P satisfying the condition $\varphi_k \subsetneq \varphi_1$ for $k = 2, \dots, n$. Then $\bigvee\limits_{2 \le k \le n} \varphi_k \subsetneq \varphi_1$ holds.*

Proof. By the condition $\varphi_k \subsetneq \varphi_1$ for $k = 2, \dots, n$, we know that $\bigvee\limits_{2 \le k \le n} \varphi_k \subseteq \varphi_1$ holds. To prove that $\bigvee\limits_{2 \le k \le n} \varphi_k$ is also a proper subset of φ_1, we choose an arbitrary query φ_j ($j \in [2, n]$) from $\{\varphi_2, \dots, \varphi_n\}$. Since φ_j is a conjunctive formula and $\varphi_j \subsetneq \varphi_1$ holds, then according to the Homomorphism Theorem [3], there must exist a homomorphism θ from φ_1 to φ_j, and two different variables x_1 and x_2 of φ_1 such that $\theta(x_1) = y$ and $\theta(x_2) = y$ hold for a variable y of φ_j. It means that, in order get $\varphi_1 - \bigvee\limits_{2 \le k \le n} \varphi_k = \emptyset$, we at least require that the results of $\bigvee\limits_{2 \le k \ne j \le n} \varphi_k$ contain the results of $(\varphi_1 \wedge x_1 \ne x_2)$. However, the domain of x_1 and x_2 has an infinite number of constants and each query φ_k ($2 \le k \ne j \le n$) can be assigned with at most a pair of different constants on x_1 and x_2. Hence, it is impossible to find such a finite set $\{\varphi_k | 2 \le k \ne j \le n\}$ of queries to satisfy $\bigvee\limits_{2 \le k \ne j \le n} \varphi_k - (\varphi_1 \wedge x_1 \ne x_2) = \emptyset$. Consequently, $\bigvee\limits_{2 \le k \le n} \varphi_k \subsetneq \varphi_1$ is proven.

Lemma 3. *Let $\Sigma_P = \{\varphi_1, \dots, \varphi_n\}$ be a finite set of queries associated with a knowledge pattern P satisfying the condition $\varphi_1 \nsubseteq \varphi_k$ for $k = 2, \dots, n$. Then $\varphi_1 \nsubseteq \bigvee\limits_{2 \le k \le n} \varphi_k$ holds.*

Proof. From a set-theoretic point of view, each φ_k $(k = 2, \dots, n)$ can be expressed as the disjoint union of two parts $\varphi_{(1,k)} = \varphi_1 \wedge \varphi_k \subsetneq \varphi_1$ and $\varphi_{(2,k)} = \varphi_k - \varphi_1$.

(1). Since $\varphi_{(1,k)} \subsetneq \varphi_1$ $(k = 2, \dots, n)$, according to Lemma 2, we have $\varphi_1 - \bigvee_{2 \le k \le n} \varphi_{(1,k)} \neq \emptyset$.

(2). Since $\varphi_1 - \varphi_{(2,k)} = \varphi_1$ $(k = 2, \dots, n)$, we have $\varphi_1 - \bigvee_{2 \le k \le n} \varphi_{(2,k)} = \varphi_1$.

Hence, we have $\varphi_1 - \bigvee_{2 \le k \le n} \varphi_k = \varphi_1 - \bigvee_{2 \le k \le n} \varphi_{(1,k)} \neq \emptyset$. $\varphi_1 \nsubseteq \bigvee_{2 \le k \le n} \varphi_k$ is proven. □

Theorem 2. *Let $P_1 = \langle \varphi, r_1 \rangle$ and $P_2 = \langle \phi, r_2 \rangle$ be two knowledge patterns over the same schema. Then $P_1 \subseteq P_2$ iff the following condition is satisfied:*

$$\begin{gathered} \forall \varphi_1 \in \Sigma^+_{P_1}.(\exists \phi_1 \in \Sigma^+_{P_2}.\varphi_1 \subseteq \phi_1 \wedge \\ \forall \phi_2 \in \Sigma^-_{P_2}.(\exists \varphi_2 \in \Sigma^-_{P_1}.(\phi_2 \wedge \varphi_1 \subseteq \varphi_2 \wedge \varphi_1))). \end{gathered}$$

Proof. Let us start with the if part. According to the first line of the condition, for each formula $\varphi_1 \in \Sigma^+_{P_1}$, there must exist a formula $\phi_1 \in \Sigma^+_{P_2}$ that contains φ_1. Then, the second line of the condition guarantees that if any results of φ_1 are eliminated by a formula ϕ_2 from P_2 then they are also eliminated by a formula φ_2 from P_1. Thus $P_1 \subseteq P_2$ holds.

For the only if part, the proof is built upon Lemmata 1 and 3. We proceed it in three steps:

First, we explain why, for each $\varphi_1 \in \Sigma^+_{P_1}$, there must exist a $\phi_1 \in \Sigma^+_{P_2}$ such that $\varphi_1 \subseteq \phi_1$. Assume that there does not exist any $\phi_1 \in \Sigma^+_{P_2}$ such that $\varphi_1 \subseteq \phi_1$ but $(\varphi_1 - \bigvee \Sigma^-_{P_1}) \subseteq (\bigvee \Sigma^+_{P_2} - \bigvee \Sigma^-_{P_2})$ holds, and then we want to prove that

$$\begin{aligned} \varphi_1 &\subseteq (\bigvee \Sigma^+_{P_2} - \bigvee \Sigma^-_{P_2}) \cup \bigvee \Sigma^-_{P_1} \\ &\subseteq \bigvee \Sigma^+_{P_2} \cup \bigvee \Sigma^-_{P_1}. \end{aligned}$$

However, because the pattern relation of P_1 is minimal, by Lemma 1, $\varphi_1 \nsubseteq \varphi_2$ holds for every formula $\varphi_2 \in \Sigma^-_{P_1}$. Furthermore, according to our assumption, we know that $\varphi_1 \nsubseteq \phi_1$ holds for every formula $\phi_1 \in \Sigma^+_{P_2}$. Since the numbers of formulae in both $\Sigma^+_{P_2}$ and $\Sigma^-_{P_1}$ are finite, by using Lemma 3, we can get $\varphi_1 \nsubseteq \bigvee \Sigma^+_{P_2} \vee \bigvee \Sigma^-_{P_1}$, which contradicts with the previous formula that is what we want to prove. Hence, we have proven that, for each formula $\varphi_1 \in \Sigma^+_{P_1}$, there must exist a formula $\phi_1 \in \Sigma^+_{P_2}$ such that $\varphi_1 \subseteq \phi_1$.

The second step is to discuss why the second line of the condition is necessary. That is, for each $\phi_2 \in \Sigma^-_{P_2}$, we need to show that there must exist a $\varphi_2 \in \Sigma^-_{P_1}$ such that $\phi_2 \wedge \varphi_1 \subseteq \varphi_2 \wedge \varphi_1$ holds. There are two sub-steps.

- We first prove that each ϕ_2 needs to satisfy $\phi_2 \wedge \varphi_1 \subseteq \bigvee \Sigma^-_{P_1}$. That is, any results of φ_1 that are eliminated by such a formula ϕ_2 in P_2 should also be eliminated by one or more formulae in P_1. Assume that there exists a ϕ_2

that does not satisfy $\phi_2 \wedge \varphi_1 \subseteq \bigvee \Sigma^-_{P_1}$. By $\phi_2 \wedge \varphi_1 \nsubseteq \bigvee \Sigma^-_{P_1}$ and the first line of the condition $\exists \phi_1 \in \Sigma^+_{P_2}.\varphi_1 \subseteq \phi_1$, there must exist some results of φ_1 that are eliminated from P_2 by ϕ_2 but still included in P_1. Thus, $P_1 \nsubseteq P_2$ and there is a contradiction.

– Second, we prove that there must exist a $\varphi_2 \in \Sigma^-_{P_1}$ such that $\phi_2 \wedge \varphi_1 \subseteq \varphi_2 \wedge \varphi_1$ holds for each $\phi_2 \in \Sigma^-_{P_2}$. Assume that there does not exist such a φ_2 satisfying $\phi_2 \wedge \varphi_1 \subseteq \varphi_2 \wedge \varphi_1$. By using Lemma 3, we would have

$$\phi_2 \wedge \varphi_1 \nsubseteq \bigvee_{\varphi_2 \in \Sigma^-_{P_1}} (\varphi_2 \wedge \varphi_1).$$

Since $\bigvee_{\varphi_2 \in \Sigma^-_{P_1}} (\varphi_2 \wedge \varphi_1) \equiv \bigvee \Sigma^-_{P_1} \wedge \varphi_1$, we then have

$$\begin{aligned} \phi_2 \wedge \varphi_1 &\nsubseteq \bigvee \Sigma^-_{P_1} \wedge \varphi_1 \\ &\nsubseteq \bigvee \Sigma^-_{P_1}. \end{aligned}$$

This contradicts with our results in the first sub-step. Hence, for each $\phi_2 \in \Sigma^-_{P_1}$, there must exist a $\varphi_2 \in \Sigma^-_{P_1}$ such that $\phi_2 \wedge \varphi_1 \subseteq \varphi_2 \wedge \varphi_1$ holds.

Our last step is to prove that each $\varphi_1 \in \Sigma^+_{P_1}$ needs to satisfy the requirements in the first two steps. Since the pattern relation of P_1 is minimal, by Lemma 1, $\varphi_1 \nsubseteq \varphi_2$ holds for any $\varphi_2 \in \Sigma^-_{P_1}$ and $\varphi_1 \nsubseteq \varphi_1^{'}$ holds for any $\varphi_1^{'} \in (\Sigma^+_{P_1} - \{\varphi_1\})$. Thus, by Lemma 3, $\varphi_1 \nsubseteq \bigvee \Sigma^-_{P_1} \vee \bigvee(\Sigma^+_{P_1} - \{\varphi_1\})$ holds. It means that each $\varphi_1 \in \Sigma^+_{P_1}$ always contributes some results into the final results of a knowledge pattern, which cannot be replaced by any other formulae in $\Sigma^+_{P_1}$. Hence, the requirements in the previous two steps need to be satisfied by each $\varphi_1 \in \Sigma^+_{P_1}$.

□

Example 5. Let us consider the knowledge patterns $P_1 = \langle \varphi_1, r_1 \rangle$ and $P_2 = \langle \varphi_2, r_2 \rangle$ shown as below.

$$\varphi_1 = \exists z_1, z_2, z_3.p_1(x, z_1) \wedge p_2(z_2, y, z_3)$$

$r_1 =$

A_{z_1}	A_{z_2}	A_{z_3}	A^*	
a	a	λ	$+$	t_1
b	b	λ	$+$	t_2
λ	λ	b	$-$	t_3

$$\varphi_2 = \exists z_1, z_3.p_1(x, z_1) \wedge p_2(z_1, y, z_3)$$

$r_2 =$

A_{z_1}	A_{z_3}	A^*	
λ	λ	$+$	t_1
λ	b	$-$	t_2

To check whether $P_1 \subseteq P_2$ holds, by Theorem 2, we just need to check whether the following containments between conjunctive queries hold, i.e., $P_1 \subseteq P_2$ holds iff (1)-(4) hold,

$$(1).\ \varphi_1^{t_1} \subseteq \varphi_2^{t_1} \qquad (2).\ \varphi_1^{t_3} \wedge \varphi_1^{t_1} \subseteq \varphi_2^{t_2} \wedge \varphi_1^{t_1}$$
$$(3).\ \varphi_1^{t_2} \subseteq \varphi_2^{t_1} \qquad (4).\ \varphi_1^{t_3} \wedge \varphi_1^{t_2} \subseteq \varphi_2^{t_2} \wedge \varphi_1^{t_2}$$

where

- $\varphi_1^{t_1} = \exists z_3.p_1(x, a) \wedge p_2(a, y, z_3)$;
- $\varphi_1^{t_2} = \exists z_3.p_1(x, b) \wedge p_2(b, y, z_3)$;
- $\varphi_1^{t_3} = \exists z_1, z_2.p_1(x, z_1) \wedge p_2(z_2, y, b)$;
- $\varphi_2^{t_1} = \exists z_1, z_3.p_1(x, z_1) \wedge p_2(z_1, y, z_3)$;
- $\varphi_2^{t_2} = \exists z_1.p_1(x, z_1) \wedge p_2(z_1, y, b)$.

Note that, pattern formulae of equivalent knowledge patterns may not necessarily be equivalent. Similarly, given two knowledge patterns $P = \langle \varphi, r \rangle$ and $P^{'} = \langle \varphi^{'}, r^{'} \rangle$ with $P \subseteq P^{'}$, we may not necessarily have $\varphi \subseteq \varphi^{'}$, e.g., $P_1 \subseteq P_2$ but $\varphi_2 \subseteq \varphi_1$ in Example 5. Nevertheless, it can be proven that, (i) if $P \subseteq P^{'}$, then either $\varphi \subseteq \varphi^{'}$ or $\varphi^{'} \subseteq \varphi$ holds; (ii) when $\varphi^{'} \subsetneq \varphi$ (i.e. $\theta(\varphi) = \varphi^{'}$ for a homomorphism θ from $sym(\varphi)$ to $sym(\varphi^{'})$), P can be transformed into $P^{''} = \langle \varphi^{'}, \theta(r) \rangle$ and $P^{'} \equiv P^{''}$ holds.

Example 6. The knowledge pattern $P_1 = \langle \varphi_1, r_1 \rangle$ in Example 5 can be transformed into $P_1^{''} = \langle \varphi_2, r_1^{'} \rangle$ where $P_1 \equiv P_1^{''}$ and $r_1^{'}$ is shown as below.

$r_1^{'} =$

A_{z_1}	A_{z_3}	A^*
a	λ	$+$
b	λ	$+$
λ	b	$-$

6 Complexity Analysis

In this section we analyse the complexity of the containment problem for knowledge patterns. There are two complexity measures to be considered – data complexity and expression complexity – which are defined in a similar sense to the ones used for relational query languages [27]. When fixing the size of the data instance upon which a knowledge model for representing identity knowledge is built, we study *expression complexity* that is the complexity of determining the containment of two knowledge patterns represented by pattern formulae and relations. The expression complexity is given as a function of the length of the representation of pattern formulae and relations or, alternatively speaking, the size of ex-queries and in-queries of knowledge patterns. In addition, we are also interested in *data complexity* that measures the computational complexity of determining containment between knowledge patterns, in which their pattern formulae and relations are fixed, as a function of the size of the data instance.

Theorem 3. *The containment problem for knowledge patterns is NP-complete with respect to expression complexity.*

Proof. Since the complexity of determining containment between conjunctive queries is known to be NP-complete [3], and determining containment of conjunctive queries is reducible to determining containment of knowledge patterns that have exactly one in-query but no ex-query, the complexity of determining containment between knowledge patterns is also NP-complete with respect to expression complexity. □

Although the expression complexity of the containment problem for knowledge patterns is NP-complete, it is measured in terms of the size of ex-queries and in-queries of knowledge patterns, which is often much smaller than the size of the data instance. We can also obtain the following result on data complexity of the containment problem for knowledge patterns.

Theorem 4. *The containment problem for knowledge patterns is in* PTIME *with respect to data complexity.*

Proof. We know that the containment problem for conjunctive queries is exactly the same problem as the query evaluation problem [3]. It is also well-known that the data complexity of evaluating conjunctive queries is in LOGSPACE [27]. Following our theorem of characterising containment of knowledge patterns (i.e., Theorem 2), we can conclude that deciding containment of knowledge patterns is in LOGSPACE and thus in polynomial time in terms of data complexity. □

7 Conclusion

The decidability and complexity of reasoning are important issues in the search for a suitable framework of representing identity knowledge. In this paper we have discussed a simple yet expressive framework for representing identity knowledge. One of the main results we obtained is that the complexity of determining containment of knowledge patterns is NP-complete in terms of expression complexity and is in PTIME in terms of data complexity. This result will lead us to develop a mechanism of finding an optimal representation of knowledge models, which can be used for improving identity knowledge management in the future.

The containment problem of knowledge patterns may also be viewed as subsumption of knowledge described by the class of queries corresponding to knowledge patterns. It thus gives rise to an interesting question on how our framework can link to the decidability and complexity of the subsumption problem in different DL languages that are combined with certain rule-based formalisms [8,13].

References

1. Credit where credit is due. Nature 462(7275), 825 (2009)
2. Fischer Identity as a Service. Architecture Overview for Client Organizations. White paper, Fischer International Identity (2009)
3. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
4. Aho, A., Sagiv, Y., Ullman, J.: Equivalences among relational expressions. SIAM Journal on Computing 8, 218 (1979)

5. Aho, A.V., Ullman, J.D.: Universality of data retrieval languages. In: Proceedings of Principles of Programming Languages, pp. 110–119. ACM (1979)
6. Arasu, A., Chaudhuri, S., Kaushik, R.: Transformation-based framework for record matching. In: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, pp. 40–49. IEEE Computer Society (2008)
7. Arasu, A., Kaushik, R.: A grammar-based entity representation framework for data cleaning. In: Proceedings of the 35th SIGMOD International Conference on Management of Data, SIGMOD 2009, pp. 233–244. ACM (2009)
8. Baader, F.: Logic-Based Knowledge Representation. In: Veloso, M.M., Wooldridge, M.J. (eds.) Artificial Intelligence Today. LNCS (LNAI), vol. 1600, pp. 13–41. Springer, Heidelberg (1999)
9. Bhattacharya, I., Getoor, L.: Deduplication and group detection using links. In: Proceedings of the 2004 ACM SIGKDD Workshop on Link Analysis and Group Detection (2004)
10. Bhattacharya, I., Getoor, L.: Iterative record linkage for cleaning and integration. In: Proceedings of the 9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, DMKD 2004, pp. 11–18. ACM (2004)
11. Calvanese, D., De Giacomo, G., Vardi, M.Y.: Decidable containment of recursive queries. Theor. Comput. Sci. 336, 33–56 (2005)
12. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, STOC 1977, pp. 77–90. ACM (1977)
13. Donini, F., Lenzerini, M., Nardi, D., Schaerf, A.: Reasoning in description logics. Principles of Knowledge Representation, 191–236 (1996)
14. Fellegi, I., Sunter, A.: A theory for record linkage. Journal of the American Statistical Association 64(328), 1183–1210 (1969)
15. Klug, A.: On conjunctive queries containing inequalities. J. ACM 35, 146–160 (1988)
16. Leclère, M., Mugnier, M.-L.: Some Algorithmic Improvements for the Containment Problem of Conjunctive Queries with Negation. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 404–418. Springer, Heidelberg (2006)
17. Levy, A.Y., Rousset, M.-C.: Verification of knowledge bases based on containment checking. Artif. Intell. 101, 227–250 (1998)
18. Nash, A., Ludäscher, B.: Processing Unions of Conjunctive Queries with Negation under Limited Access Patterns. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 422–440. Springer, Heidelberg (2004)
19. Newcombe, H., Kennedy, J., Axford, S., James, A.: Automatic linkage of vital records. Science 130(3381), 954–959 (1959)
20. Pasula, H., Marthi, B., Milch, B., Russell, S., Shpitser, I.: Identity uncertainty and citation matching. In: NIPS. MIT Press (2003)
21. Sagiv, Y., Yannakakis, M.: Equivalences among relational expressions with the union and difference operators. Journal of the ACM 27, 633–655 (1980)
22. Sarawagi, S., Bhamidipaty, A.: Interactive deduplication using active learning. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2002, pp. 269–278. ACM (2002)
23. Shmueli, O.: Equivalence of datalog queries is undecidable. J. Log. Program. 15, 231–241 (1993)
24. Singla, P., Domingos, P.: Object Identification with Attribute-Mediated Dependences. In: Jorge, A.M., Torgo, L., Brazdil, P.B., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, pp. 297–308. Springer, Heidelberg (2005)

25. Tejada, S., Knoblock, C.A., Minton, S.: Learning object identification rules for information integration. Information Systems 26 (2001)
26. Ullman, J.D.: Information integration using logical views. Theor. Comput. Sci. 239, 189–210 (2000)
27. Vardi, M.: The complexity of relational query languages. In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, pp. 137–146. ACM (1982)
28. Wei, F., Lausen, G.: Containment of Conjunctive Queries with Safe Negation. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572, pp. 346–360. Springer, Heidelberg (2002)

Privacy Preserving Mining Maximal Frequent Patterns in Transactional Databases

Md. Rezaul Karim[1], Md. Mamunur Rashid[1], Byeong-Soo Jeong[1], and Ho-Jin Choi[2]

[1] Dept. of Computer Engineering, Kyung Hee University
1- Seochun-dong, Kiheung-gu, Yongin-si, Kyunggi-do 446-701, Republic of Korea
[2] Computer Science Dept., Korea Advanced Institute of Science and Technology
335-Guseong-dong, Yuseong-gu, Daejeon 305-701, Republic of Korea
{asif_karim,mamun,jeong}@khu.ac.kr, hojinc@kaist.ac.kr

Abstract. Problem of finding frequent patterns has long been studied because it is very essential to data mining tasks such as association rule analysis, clustering, and classification analysis. Privacy preserving data mining is another important issue for this domain since most users do not want their private information to leak out. In this paper, we proposed an efficient approach for mining maximal frequent patterns from a large transactional database with privacy preserving capability. As for privacy preserving, we utilized prime number based data transformation method. We also developed a noble algorithm for mining maximal frequent patterns based on lattice structure. Extensive performance analysis shows the effectiveness of our approach.

Keywords: privacy preserving data mining, maximal frequent pattern, prime number theory.

1 Introduction

Finding frequent patterns plays an important role in all data mining tasks such as association analysis, clustering, and classification. Generally, since the set of maximal frequent patterns is much smaller than the set of frequent patterns in a large transactional database, it is very helpful if we can find maximal frequent patterns efficiently. Thus, developing efficient maximal frequent pattern mining techniques has been an important research direction in data mining.

At the same time, the privacy is another concern in mining business oriented data since most companies may not want their business information to be exposed to the public. Thus, when data mining tasks are performed by the third party vendor it is necessary to hide their business oriented transaction information from rival companies.

There have been a lot of research works to deal with finding maximal frequent patterns from a transaction database and enforcing information security properly during data mining. However, to the best of our knowledge, there has been no work to deal with these problems together. In this paper, we propose an efficient approach for mining maximal frequent patterns with privacy preserving capability. In order to hide

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 303–319, 2012.
© Springer-Verlag Berlin Heidelberg 2012

transaction information from outside, we encrypt transaction information by using prime number based transformation. Our proposed mining algorithm is designed to use transformed database instead of using original transaction database.

Main contributions of this paper are as follows: (1) We proposed a simple but very effective approach for finding maximal frequent patterns with privacy preserving capability. (2) For this, we devise prime number based data transformation and provide a way to construct lattice structure and traverse it from the transformed database. (3) We also provide an extensive performance study to show the effectiveness of our approach.

The rest of this paper is organized as follows. Section 2 briefly describes related works. In Section 3, we described background study and the problem definition. We explained proposed approach for mining maximal frequent patterns from a transformed database in Section 4. Section 5 summarizes the experimental results. Finally, we conclude in Section 6.

In this paper we will use 'itemset' and 'pattern'; 'descendent' and 'child'; 'transaction vector' and 'transaction value' interchangeably. We used acronym TPFPM for Third Party Frequent Pattern Miner and MFPM for Maximal Frequent Pattern Miner. And we used the generic name lattice to indicate our proposed lattice structure.

2 Related Works

In order to preserve the privacy of the client in data mining process, a variety of techniques based on random perturbation of data records have been proposed recently [14]. In this literature two dominant methods Randomization and Distortion are provided as a means for privacy preserving. But the normal distortion procedure does not provide the flexibility of tuning the probability parameters for balancing privacy and accuracy parameters, and each item's presence/absence is modified with an equal probability conventional wisdom held that data mining, with its promise to efficiently discover valuable, non-obvious information from large databases, is particularly vulnerable to misuse [3]. Literature [4] predicted the making of a conflict between data mining and privacy. The objective of data mining is to generalize across populations, rather than reveal information about individuals.

The problem of mining MFP was first proposed by Bayado in [11] particularly, MFP can derive all frequent itemsets effectively, so many algorithms mining MFP are proposed at literatures [2], [5], [8], [9], [17]. Most of the algorithms for mining MFP are similar to the Apriori [7] in a bottom-up breadth-first fashion. MaxMiner [11] is a typical algorithm to mine the MFP, which extends the Apriori algorithm. It builds a concept frame of Rymon's set enumeration tree [18], and uses a breadth-first traversal of the search space. Though superset frequency pruning reduces the search time drastically, MaxMiner still needs many passes to get all MFP.

Pincer-Search [13] combines both the bottom-up and the up-bottom searches to find the maximal frequent itemsets. In general, if some maximal frequent itemsets are long and the maximal frequent itemsets are distributed in a scattered manner, then the problem of discovering the MFS can be very hard. In this case, even Pincer-Search might not be able to solve this problem.

In [10], a depth-first algorithm MAFIA is proposed. It uses a bitmap representation and a linked list to organize all frequent itemsets and has lots of candidate generation problem and hence is inefficient to mine MFPs. Similar to MaxMiner, GenMax [12] is another depth-first algorithm using Rymon's set enumeration technique [18]. Its main contribution is "progressive focusing" technique, and diffset propagation to perform fast frequency computation to maintains a set of local MFP which compares with the newly founded frequent itemsets to reduce the cost of subset tests. But although it is able to reduce the candidate sets but it was found that the GenMax spends half of its time in maximality checking; it degrades the performance.

PC_Miner algorithm [2] is based on prime number characteristic, and proposes a novel tree structure called PC_Tree. The algorithm uses a prime-based database encoding technique, which can reduce the size of transaction database efficiently. It first transformed every transaction into a compact value called TV (Transformed Vector) value and then from the transformed vector database mines the set of maximal frequent patterns. It shows better compaction rate and mining time; but it has some limitations. Firstly it has parent – child reordering problem. Secondly there is redundancy in TV transformation technique because the PC_Tree algorithm assigns and transforms each transaction including infrequent items so the compression rate is not satisfactory. Thirdly to construct the PC_Tree they sorted the transformed database in ascending order of transformed vector value, since the sorting cost is not a trivial one, it creates an extra overhead in terms of time and memory usage. Fourthly, it takes more time to discover maximal frequent patterns because it first generates the head sets to generate frequent patterns and later combines these frequent patterns to discover the maximal frequent patterns.

Most recently literature [11] was proposed to mine the set of maximal frequent itemsets using two steps: in the first place, they compressed the large database into a condensed, smaller data structure through dividing the attributes, called information matrix which can avoid repeated, costly database scans. Secondly, the maximum frequent itemsets is generated ultimately be means of cover relation using intonation matrix and the costly generation of a large number of candidates is avoided. In such case, both I/O time and CPU time are reduced by eliminating the candidates that are subsets of the maximal frequent itemsets. But it also has a drawback since the database is divided into three groups along column direction. So if the number of items in every group is n, then it needs to allocate a matrix with size 2^n x 2^n.That means if the size of original database is increases; the matrix size is also increases exponentially and performance degrades because any matrix based representation is not so efficient with O (n^2) complexity.

3 Problem Definition

In this section we will first define the problem of maximal frequent pattern mining and then present some preliminary knowledge that is necessary to understand our work.

3.1 Maximal Frequent Pattern Mining Problem

Let a set of distinct items $I = \{i_1, i_2, \ldots, i_n\}$ and n is the number of distinct items. We assume that items are ordered by certain predefined way. Transaction t is an ordered list of items, denoted as $t = [i_1\ i_2, .. \ i_m]$ where $i_1 < i_2 < \ldots < i_m,$ and $m \leq n$.

Table 1. A Transactional Database

TID	Items
1	A, B, C, D, F
2	A, B, C, E
3	B, C, D, E, F
4	A, C, D, E
5	C, D, F
6	D, E, F
7	D, E
8	C, D, F
9	C, F
10	A, C, D, E
11	C, E

A transactional database $T = \{t_1, t_2...t_N\}$ is a set of N transactions and $|N|$ is the number of total transactions. A set $X \subseteq I$ is called a pattern. If $X \subseteq t$, it is said that X occurs in t or t contains X. *Support(X)* denotes the percentage of transactions that contain X. If *Support(X)* ≥ *min_sup* we say that X is a frequent pattern. If X is a frequent pattern and no superset of X is frequent, we say that X is a maximal frequent pattern.

For example, in Table 1, the occurrences of patterns "*CD*", "*DE*" and "*CDF*" are 6, 5 and 4 respectively. If we assume that *min_sup* is 4, all of these are frequent patterns. But, "*CD*" is not a maximal frequent pattern because its super pattern "*CDF*" is also a frequent pattern. The problem of mining maximal frequent patterns is to find all maximal frequent patterns whose *support* is no less than a user-given *min_sup* threshold.

3.2 Privacy Preserving Frequent Pattern Mining

To preserve the privacy it is necessary to transform all the transactions in a compacted layout using some suitable transformation techniques so that original transaction is hided from any third party frequent pattern miner. Suppose we have a transactional database DB and DB^* be a transformed database from the original database DB. It is this transformed database DB^* that is supplied to the third party data miner, along with the minimum support threshold value. The data miner mines the transformed database DB^* to estimate the frequent patterns with support count satisfying the minimal support in the original database DB without knowing the sensitive transaction information.

4 Proposed Approach

4.1 The Privacy Preserving Framework

As depicted in Fig. 1, our framework encompasses a transactional database. The data owner or organization process the database to modeled into a TV database DB^* (Transaction vector database), the internal processing unit has a unit called Encoder which encodes the original transactional information into corresponding transactional vector database DB^*, and a decoding tools called Decoder which decodes the mined values into maximal frequent pattern form. Besides there are three third party tools called i) The lattice construction tools (Prime-based lattice structure construction tools), ii) MFPM algorithm tools and ii) A temporary buffer to hold the maximal TV values and these values are handover to the data owner. The whole process goes as follows:

(i) First of all the owner gives this transformed database to any third party frequent pattern miner. (ii) The third party lattice construction tools create and insert the whole TVs into a lattice structure. (iii) After that it mines the maximal TVs by traversing the lattice structure using MFPM algorithm tools and stores these TV values in a temporary buffer. Since the TPFPM has no decoding tools so the privacy will not be violated, and it is guaranteed that the privacy will be preserved (iv) Upon request from the data owner the TPFMP hand over the maximal TV values to the Decoder tolls (v) Finally the Decoder factorizes the maximal TV values to its corresponding prime factor and retrieves the original maximal frequent patterns.

4.2 Database Transformation and Decoding Technique

In this section, we described the data transformation technique in more details. In fact the TV database is a numerical encoder which hides the underlying transactions information and basically provides the data privacy. Compactness is desirable in all sorts of algorithms and obviously, reducing of the size of database can enhance performance of mining algorithms. To hide the original transactional database information, we use prime number based transformation.

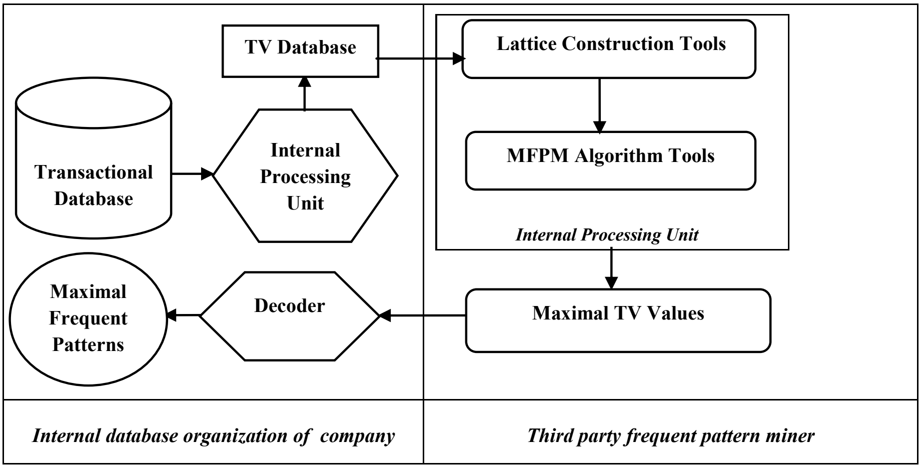

TV Database
Lattice Construction Tools
Transactional Database
Internal Processing Unit
MFPM Algorithm Tools
Internal Processing Unit
Maximal Frequent Patterns
Decoder
Maximal TV Values
Internal database organization of company
Third party frequent pattern miner

Fig. 1. The Proposed Framework for Mining Privacy Preserving Maximal Frequent Patterns

In prime number theory, it is well known that a positive integer N can be expressed by unique product of prime numbers, that is $N = P_1^{m1} * P_2^{m2} * \ldots * P_r^{mr}$ where P_i is prime number, $P_1 < P_2 < \ldots < P_r$ and m_i is a positive integer. As for prime numbers, *2, 3, 5, 7…* we map these prime numbers to each item in a transactional database. For example, we map *2* as "*A*", *3* as "*B*", *5* as "*C*", and so on. Then we express each transaction as product of mapped prime numbers. In the case that we transform a transaction into $P_1^{m1} * P_2^{m2} * \ldots * P_r^{mr}$, where m_i will be always *1* because there is no duplicate item in a transaction. Table 2 shows this transformation. Here in Table 2, we eliminate infrequent one item "*B*" in a transaction for reducing search space as like FP-tree structure [7].

For each transaction, transformed value (TV) is obtained by multiplying every prime number in transformed transaction. For example, transformation of transaction *1* would be {*2, 3, 5, 11*} and its transformed value is $2*3*5*11 = 330$. Now we will describe some interesting but very useful properties of prime based data transformation techniques.

Property 1: If two transactions are different (i.e., they have different items), then their transformed values are also different. By definition of transformed transaction, if transformed values of transactions are the same, they may be represented as same $P_1 * P_2 * \ldots * P_r$ that means same transaction.

Property 2: Let m be a greatest common divisor (gcd) of transformed values of two transactions, t_1 and t_2. We know that m is represented by $P_1 * P_2 * \ldots * P_k$ where $k \geq 1$. By definition of transformed transaction, t_1's TV_1 is $(P_1 * P_2 * \ldots * P_k) * P * \ldots * P_j$ (where $k \geq 1$ and $i \leq j$) and t_2's TV_2 is $(P_1 * P_2 * \ldots * P_k) * P_m \ldots * P_n$ (where $k \geq 1$ and $m \leq n$). Thus, t_1 and t_2 has common items mapped to P_1, P_2, and P_k.

Property 3: If TV_1 of t_1 is divided by TV_2 of t_2 then t_1 is a super pattern of t_2. By definition transformed transaction, TV_1 would be $(P_1 * P_2 * \ldots * P_k) * P_i * \ldots * P_j$ (where $TV_2 = (P_1 * P_2 * \ldots * P_k)$ and $i \leq j$)) It means that t_1 has items mapped with P_1, P_2, and P_k which are same items of t_2. Thus, t_1 is a super pattern of t_2. In the same way, we know that t_2 is a sub pattern of t_1.

Table 2. A Transformed Transactional Database

TID	Transaction	Transformation	TV
1	A, C, D, F	2, 3,5,11	330
2	A, C, E	2, 3,7	42
3	C, D, E, F	3,5,7,11	1155
4	A, C, D, E	2, 3, 5,7	210
5	C, D, F	3, 5,11	165
6	D, E, F	5,7,11	385
7	D, E	5,7	35
8	C, D, F	3,5,11	165
9	C, F	3,11	33
10	A, C, D, E	2,3,5,7	210
11	C, E	3,7	21

So important procedures are done by only three simple mathematical operations product, division and greatest common divisor. And obviously using mathematical operation enhances the performance instead of string operation. As depicted in Figure 1, we transform our original transactional database in table-1 into *DB** which has transformed transaction values and disclose it to third party vendor for data mining.st Since this *DB** has same information with original transactional database, third party vendor can correctly find maximal TV value (As maximal frequent patterns) without noticing that which items exist in a maximal frequent pattern. From the characteristics of the prime number it is clear that a decomposition of a number n into (finitely many) prime factors p_1, p_2, ... to p_t is called prime factorization of n. As in this example, the same prime factor may occur multiple times.

$$n = p_1 * p_2 * ... * p_t$$

The fundamental theorem of arithmetic can be rephrased so as to say that any factorization into primes will be identical except for the order of the factors. After finding all maximal transformed values, we decode them into a set of items inside company organization. Then it retrieves the original maximal frequent patterns and store into the local disk.

4.3 Mining Maximal Frequent Pattern with Privacy Preserving

In this section, we will describe the procedure which finds maximal transformed value from the transformed database. First we will describe the lattice construction algorithm, after that we will describe the MFPM algorithm for finding the set of maximal TV values. Finally we will explain a step by step example for finding the set of maximal frequent patterns from the original database.

4.3.1 The Lattice Construction Algorithm

The lattice structure includes a root and some nodes that formed sub lattice structures as children of the root or descendants. The node structure consisted with mainly of several different fields: value, local-count, global-count, status and link. The value field stores TV to records which transaction represented by this node. The local-count field set by 1 during inserting current TV and it is increased by 1 if its TV and current TV are equal. The global-count field registers support of a pattern which presented by its TV. In fact during of insertion procedure the support of all frequent and infrequent patterns is registered in the global-count field. The status field is to keep track of traversing. When a node is visited in the traversing procedure the status field is changed from 0 to 1. The link field is to form sub lattice structures or descendants of the root. Figure 2 has shown the pseudo code for the lattice construction algorithm. TV of the root is assuming to be NULL and can be divided by all TVs. First we check the divisibility between first two TVs. If the first one is divided by second TV or vice-versa then the dividend will be the parent node and the divisor will be the descendent node, otherwise we just insert these two TVs as two unique nodes. Below the insertion rules from 1 to 5 will explain how to insert into the lattice structure. The lattice construction operation mainly consists of insertion procedure and re-ordering that inserts TV(s) into the lattice, based on the five rules below: We will explain every rule with suitable examples.

Rule- 1: If TV of node n_r is not a divisor or dividend of any existing node in the lattice structure then just insert it as a new node and its local count will be 1.

For example suppose two TV values are 1155 and 330 then obviously 1155 is not divided by 330 and vice versa, so 1155 and 330 will be inserted as a unique node.

The Lattice Construction Algorithm

Input: A transactional database and a minimum support threshold δ.

Output: Lattice-the complete set of transactions transformed into transaction vectors and presented by nodes of the lattice.

Parameters: The LATTICE_STRUCTURE (TV_i) procedure creates and inserts into the lattice.TV_i is the transaction vector of corresponding transaction N_i, node is a lattice node. Also N indicates the total number of transactions in the database.

```
1  Begin
2   Scan the transactional database DB and remove infrequent
    1-itemsets and construct the transformed database DB*//
    inside the internal processing unit it goes as follows:
     for (i=0; i<N; i++)
       2.1    Assign a unique prime number to each frequent
              item in the database according to the
              ascending order of support.
       2.2    Transform each transaction into its
              corresponding TV value and construct
              transformed database DB*.//Upon request from
              the TPFPM, the data owner handover it to the
              TPFPM.
3   Scan the transformed database DB* once and insert TV
    values into the lattice //Inside the processing unit of
    TPFPM.
     for (i=0; i<ΣTV_i; i++)
       3.1  Create the root of the lattice structure and
            label it as NULL.//It is divisible by every TV
            value in the DB*.
       3.2  Insert TVs into the lattice based on mentioned
            rules 1, 2, 3,4 and 5.Call procedure
            LATTICE_STRUCTURE (TV_i) to insert.
4   End
```

Fig. 2. The Lattice Construction Algorithm

Rule-2: If TV of node n_r and n_s is equal then $r = s$. Insertion procedure will increases local-count field of node n_r by 1 if the current TV is equal with TV of n_r.

For example suppose TV value 210 already inserted a into the lattice structure and if there is another entry of 210 in the database then the local count of 210 will be 1+1=2.

Rule- 3: If TV of node n_r is a divisor of more than one TV then, global count of this node will be equal to how many dividend TVs of n_r are there in DB^*.

For example TV 35 which is a divisor of node 210 and 385 at the same time, then global_count of node 35 will be 2 and insert it below smaller TV value.

Rule-4: If a node n_r is already inserted and node n_s is going to be inserted and i) if n_s is divided by n_r and ii) if n_r> n_r; then re-order n_r and n_s (i. e. n_s will be the parent node and n_r will be the descendent node). For example suppose TV value 42 is already inserted in the lattice and 210 is going to be inserted then obviously 210> 42 and 210 is divided by 42 so we re-order these two values. Hence 210 will be the parent node and 42 will be the child node.

Rule-5: Root $R= (n_i , n_{i-1}..., n_j, Root)$ is a descendant iff TV of node $n_r \in R$ $(i \leq r \leq j)$ can divide all TVs kept in nodes $R_r= (n_{r+1},n_{r+2},\ldots.n_j,Root)$. For example suppose we have three TVs 210, 42 and 21 then 210 is the root of every node in the lattice structure. Because 210 is divisible by 42 and 21.

Moreover we repeatedly update the connection link from parent node to descendent nodes. So if a node's global count is n then the number of incoming link will be (n-1). So from the lattice structure we can observe both the local as well as global count clearly and there is no ambiguity like [2].

4.3.2 The MFPM Algorithm

The MFPM algorithm is uses to traverse the lattice structure to find out the set of maximal TVs to generate maximal frequent patterns by the decoder. As explained in previous section, during of insertion each TV in the lattice, the following procedures are done.

a) Local-counting.
b) Global-counting.
c) Link updating
d) Item re-ordering

The MFPM algorithm traverses the complete lattice structure to discover the maximal TV values in top-down fashion. There is no need to scan the database again, because all the information about items and patterns are stored in the lattice. We will present a short description of our proposed MFPM algorithm rather than a pseudo code notation.

To mine the maximal TV values from the lattice structure; we traverse it from the root to each branch of the lattice. First it find out the list '*L*' of nodes that have global count greater than minimum support threshold, since these nodes hold a transaction value so it is more likely to be a candidate of maximal frequent patterns. And it is clear that if the TV value of a node n_r is included in the candidate list then all of its child nodes are divisor so these child values are pruned even if the child nodes are frequent.

Table 3. *DB** database

TV
330
42
1155
210
165
385
35
165
33
210
21

Now taking each branch as a new entry to an array list we find the set of TV values for a given node. Taking the first node of first branch and the first, second and third node of second branch as the array list we find the greatest common divisor (property 2) of these four nodes as '*G*'. We include this greatest common divisor values in the list '*L*'. By this way we can reduce the search space. If the TV value of the first node of a branch (or TV value of any other node) already included in the list *L* then we exclude this node.

After that the decoder factorizes these maximal TV values into prime's product and retrieves the original maximal frequent patterns. Now we will explain our mining task using a practical example.

4.3.3 Example

As described in section 4.2 infrequent items are already removed from the database DB to reduce the search space as well as transformed database size. The reduced DB has been shown in table-2, for the maximal frequent pattern mining.

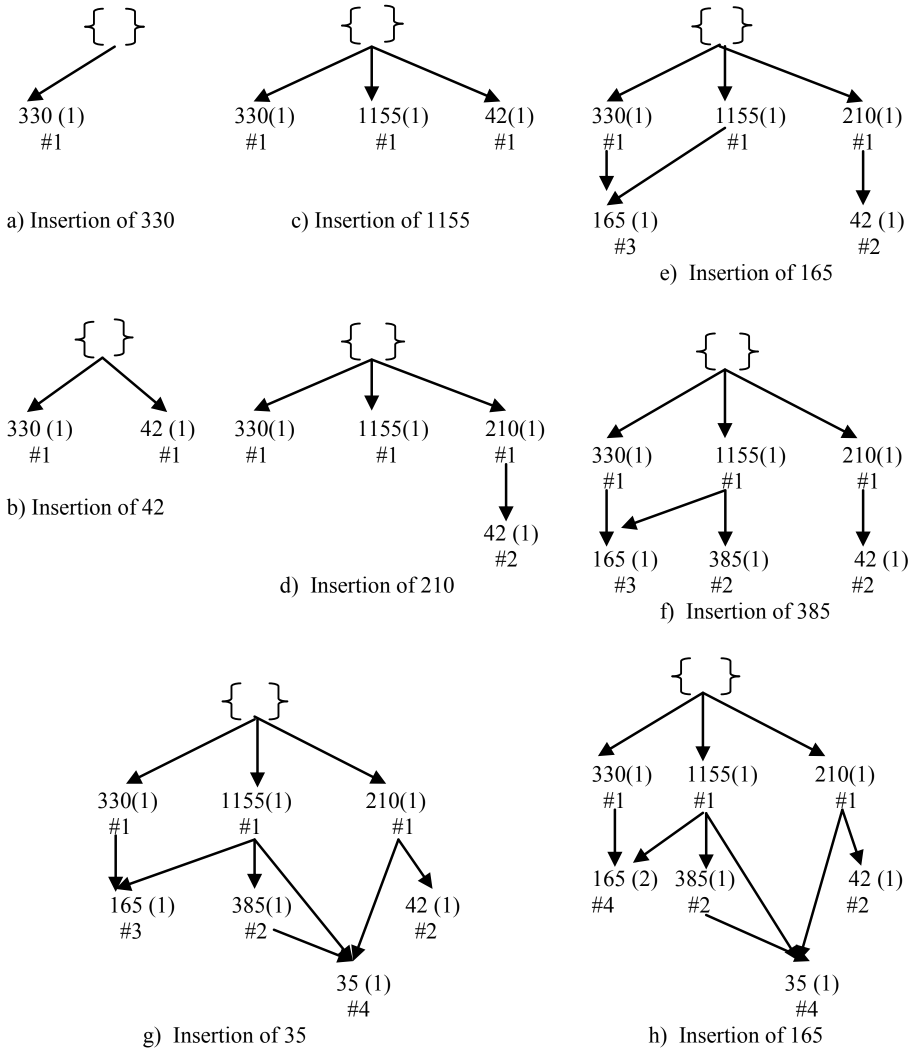
Now we will describe the creation and insertion procedure into our proposed lattice structure. First we take 330 and 42; 330 is not divided by 42 so these two values will be inserted as two new nodes. TV value 1155 also inserted as unique node. Next 210 is the dividend of 42 and 42< 210, so we re-order them; 210 will be parent node and 42 will be the descendent node; we update the link accordingly.

TV value 165 is a divisor of both 330 and 1155 but 330<1155, so 165 is inserted as the descendent node of 330 and the global count of 165 will be 3 and we update the link accordingly. Next TV value 385 is the divisor of 1155 so it will be the descendent node of 1155 and global count will be 2 as well. TV value 35 is the divisor of 210, 1155 and 385 at the same time but among them 210 is the smallest so it is inserted as the descendent node of 210 and the link and global count will be updated accordingly.

Next TV value 165 is already inserted in the lattice so we just increase the local count and global count of the node 165. At the same time we update the global count of all divisors of 165 as well. TV value 33 will be inserted as the descendent of 165 although it has another two dividend node 330 and 1155. Same rule will be applied for next TV value 210.

Finally 21 will be inserted as the descendent node of 42, although it has three dividend node 1155, 42 and 210 but 42 is the smallest.

Figure in the next page shown step by step the proposed lattice structure construction procedure. Here (...) indicates the local count of a TV value; #... Indicates the corresponding global count of a TV value and the arrow indicates the dividend-divisor relationship between nodes of the lattice structure.



a) Insertion of 330

c) Insertion of 1155

e) Insertion of 165

b) Insertion of 42

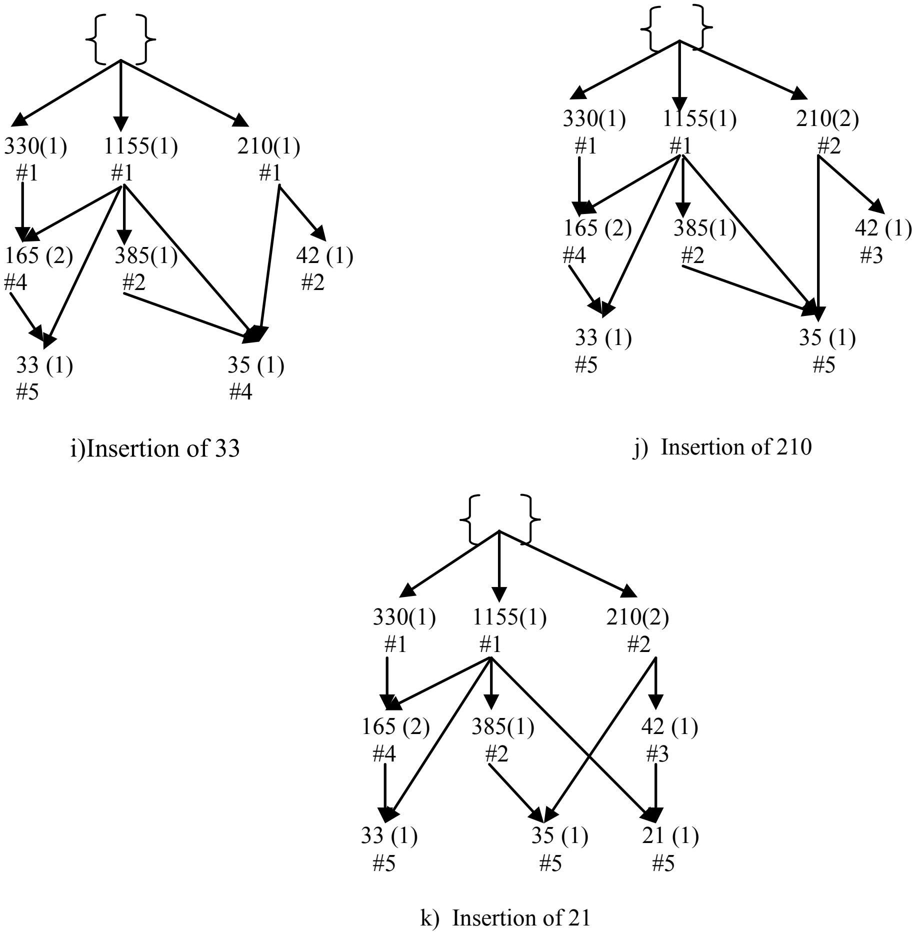
d) Insertion of 210

f) Insertion of 385

g) Insertion of 35

h) Insertion of 165

Fig. 3. Step by step lattice construction procedure. Figure 3(a) - 3(h).



i)Insertion of 33

j) Insertion of 210

k) Insertion of 21

Fig. 3. Step by step lattice construction procedure. Figure 3(i) - 3(k).

4.3.4 Mining Maximal Frequent Patterns from the Lattice Structure

To mine the maximal TV values from the lattice in figure-3, we traverse from root to each branch of the lattice structure. We maintain a list of maximal frequent TV values called 'L' and traversing the lattice structure we get the following list of TVs that satisfy minimum support threshold, *L*= {165, 35, 21}.

Although 35 satisfies the minimum support threshold, TV value 35 is the divisor of 165; so 35 has been excluded and pruned from the lattice and the list *L.* Now taking each branch as a new entry to an array list we find the set of TVs calculated previously for a given node. Now we find the products of the greatest common divisors of the candidate paths. Table 5 has shown the resulting value calculations. G value 21 and 35 are already included in the list *L* so these values will not be included in the list *L.* G value 6 is the unique so included 6 in the list. So the final list will be *L*= {165, 35, 21, 6}.

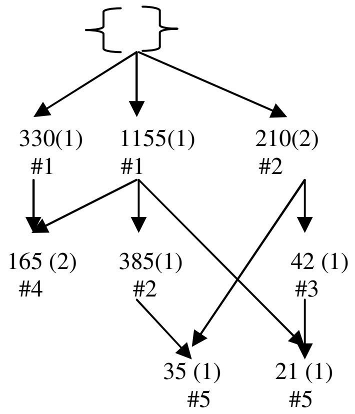


Fig. 4. The Pruned Lattice with Global and Local count (inside brackets local count, # indicates global count)

Finally the third party tools MFMP hand over the set of maximal TVs to the Decoder of the data owner. Then the Decoder converts these values into corresponding primes factors. Table-6 has shown the frequent patterns retrieval techniques. So finally we get four maximal frequent patterns; which are as follows: {C, E}, {D, E} and {C, D, F} and {A, C}.

Table 5. Candidate GCD value list for Maximal TV values: *L*

Path	TV list	Greatest Common Divisor (*G*)
P_1	1155, 210, 210, 42	21
P_2	385,1155, 210, 210	35
P_3	330, 210, 210, 42	6

Table 6. Retrieval of Maximal Frequent Patterns from Maximal TV values

Maximal TV values	Prime Factorization	Maximal Frequent Patterns	Support
165	3*5*11	C, D, F	4
35	5*7	D, E	5
21	3*7	C, E	5
6	2*3	A, C	4

5 Experimental Results

In this section, we evaluate the performance of our method. All programs were written and compiled using Microsoft Visual C++6.0 and running with the Microsoft Windows XP operating system with Pentium D 2.13 GHz processor and 2 GB of main memory. In the first experiment we used synthetic sparse datasets T10I4D100K generated by the program developed at IBM Almaden Research Center and real dense Kosarak datasets. The number of transactions, the average transaction length and the average pattern length of T10I4D100k are set to be 100K, 10 and 4 respectively. The Kosarak dataset records consist of the characteristics of various Kosarak species. The number of records, the number of items and the average record length are set to be 900002, 41270 and 8.1 respectively. We consider δ % of this dataset where δ will be increased from 1 to 10 to evaluate how much the data transformation technique can compact the size of the dataset. Fig.5 shows comparison of the size of original dataset with the size of transformed dataset using our data transformation technique. It indicates that the compaction size is more than 50%. Obviously, the compactness rate for real data can be more than synthetic data used in the experiments. This is because; the size of the TV used for a transaction is almost independent of kind of dataset, but the average length of items in real datasets is bigger than in synthetic dataset [2]. And previous experiments [2] showed that by applying this data transformation technique, the size of real transaction database can be reduced more than half. But using our approach the reduction rate of the original database is higher than [2] because of 1-itemset filtering.

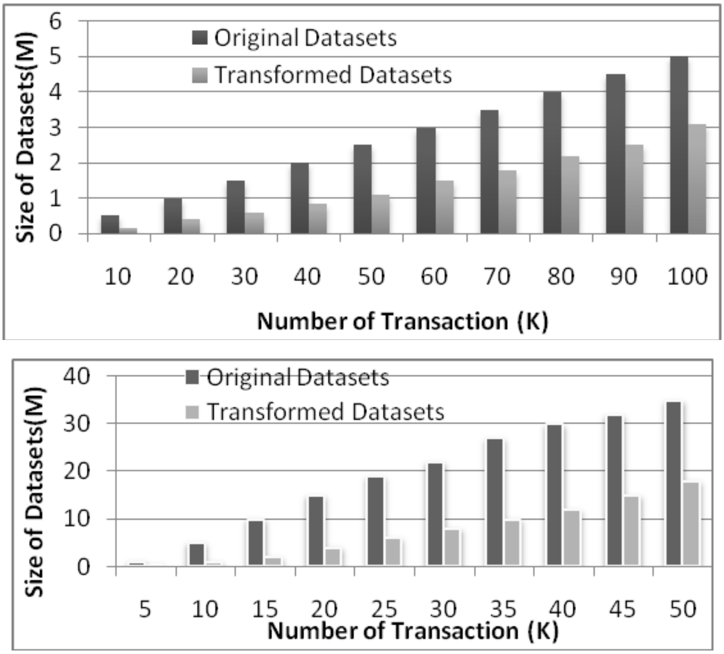


Fig. 5. Compactness of data transformation. Above: synthetic dataset, below: Kosarak dataset.

In second experiment, we compared the construction time and memory usage among the lattice structure, PC_Tree [2] and FP-tree structure. From fig-6 we can observe that the lattice shows better runtime because of the support pruning and size reduction of the datasets also the lattice structure consumes less memory space because of (i) more compaction in the datasets and (ii) it does not need the transformed TV database sorting. Actually the sorting and compaction rate degrades the overall performance of PC_Tree [2].

In third experiment we compared the performance of PC_Miner [2], MAFIA [9], GenMax [11] and MFPM. Fig-7 indicates that MFPM outperforms PC_Miner, MAFIA and GenMax because PC_Miner, MAFIA, and GenMax has relatively high candidate generation problem which is an obstacle of Apriori based pattern mining.

But MFPM only checks the nodes that has support more than minimum support threshold and performs some greatest common divisor operations on complete pruned lattice structure; and usually mathematical operations takes negligible times. Also MFPM uses the list structure to hold the candidate maximal TV values and any list based representation can be build in linear time.

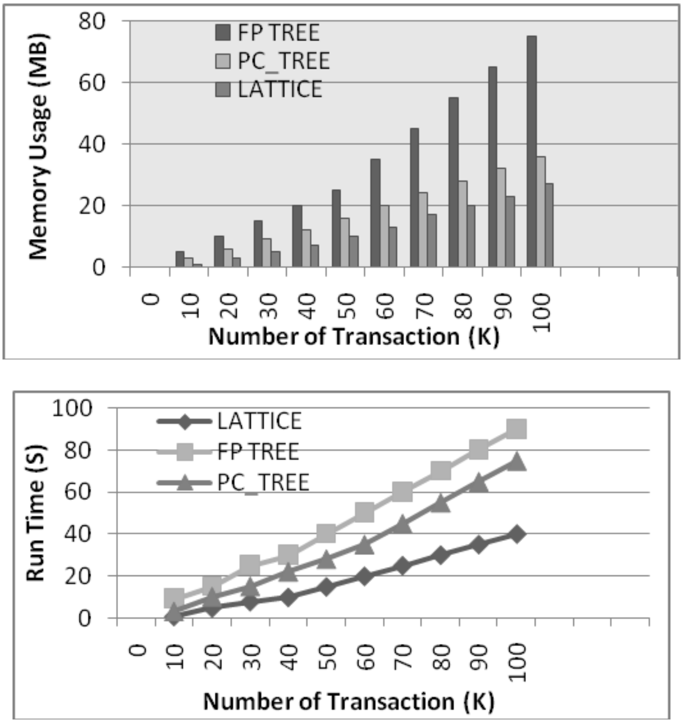


Fig. 6. Performance comparison among FP-Tree, PC_Tree and lattice; above: memory usage; below: runtime comparison on Kosarak dataset

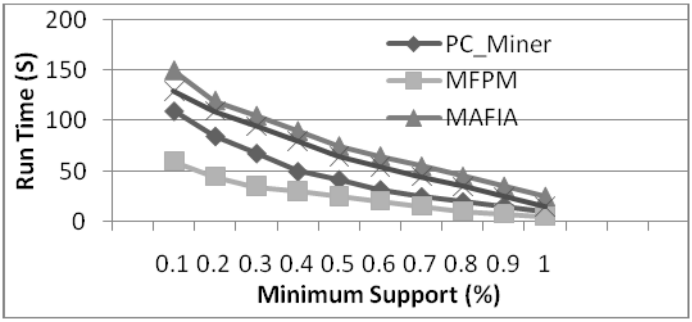


Fig. 7. Runtime comparison of MFPM vs. MaxMiner, PC_Miner, GenMax and MAFIA on Kosarak dataset

6 Conclusion

In this paper, we proposed a numerical method to mine maximal frequent patterns with privacy preserving capability. Our method showed an efficient data transformation technique, a novel encoded and compressed lattice structure and MFPM algorithm. The proposed lattice structure and MFPM algorithm reduces both search space as well as searching time. The experimental results showed that MFPM algorithm outperforms PC_Miner and existing maximal frequent pattern mining algorithms. Besides the lattice structure outperforms FP- like tree and PC_tree algorithm as well.

Acknowledgements. This work was supported by a grant from the NIPA (National IT Industry Promotion Agency, Korea) in 2012 (Fostering Global IT Human Resource Project) and the National Research Foundation (NRF) grant (No. 2011-0018264) of Ministry of Education, Science and Technology (MEST) of Korea.

References

1. Wang, J.L., Xu, C.F., Pan, Y.H.: An Algorithm for Mining Privacy-Preserving Frequent Itemsets. In: International Conference on Machine Learning and Cybernetics (2006)
2. Mustapha, N., Hossein, M., Shahraki, N., Mamat, A.B., Sulaiman, M.N.B.: A Numerical Method for Frequent Patterns Mining. Journal of Theoretical and Applied Information Technology
3. Fu, A.W.C., Wang, K.: Privacy-Preserving Frequent Pattern Mining Across Private Databases. In: Proceedings of the Fifth IEEE International Conference on Data Mining, ICDM 2005 (2005)
4. Oliveira, S.R.M., Zane, O.R.: Privacy Preserving Frequent Itemset Mining. In: IEEE International Conference on Data Mining Workshop on Privacy, Security, Maebashi City, Japan
5. Zaki, M.J.: Scalable Algorithms for Association Mining. IEEE Transactions on Knowledge and Data Engineering (2000)
6. http://en.wikipedia.org/wiki/Prime_number

7. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceeding of the 20th Inl. Conf. on Very Large Databases (VLDB), Santiago, Chile, pp. 487–499 (1994)
8. Gouda, K., Zaki, M.J.: Efficiently Mining Maximal Frequent Itemsets. In: Proc. of the 1st IEEE International Conf. on Data Mining, San Jose, USA, pp. 163–170 (2001)
9. Burdick, D., Calimlim, M., Gehrke, J.: MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. In: Proceedings of the 17th Inl. Conf. on Data Engineering, Germany (2001)
10. Gouda, K., Zaki, M.: A New Method for Mining Maximal Frequent Itemsets. In: Proceedings of the World Congress on Engineering, London, U.K (2008)
11. Burdick, D., Calimlim, M., Gehrke, J.: GenMax: An Efficient Algorithm for Mining Maximal Frequent Itemsets. In: Data Mining and Knowledge Discovery, The Netherlands (2005)
12. Mustapha, N., Sulaiman, M.N., Othman, M., Selamat, M.H.: Fast Discovery of Long Patterns for Association Rules. International Journal of Computer Mathematics (2003)
13. Shrivastava, R., Awasthy, R., Solanki, B.: New Improved Algorithm for Mining Privacy Preserving Frequent Itemsets. International Journal of Computer Science & Informatics 1 (2011)
14. `http://empslocal.ex.ac.uk/people/staff/mrwatkin/zeta/tutorial.htm`
15. Wang, H., Hu, C., Chen, Y.: Mining Maximal Patterns Based on Improved FP-lattice structure and Array Technique. In: 2nd International Conference on Future Computer and Communication (2010)
16. Leung, C.K., Khan, Q.I., Hoque, T.: CanTree: A canonical-order tree structure for frequent pattern mining. Knowledge and Information Systems (2007)
17. Kai, Y., Yuan, M.: A Fast Algorithm For Discovering Maximum Frequent Item sets. In: IEEE 3rd International Conference on Communication Software and Networks, ICCSN (2011)
18. Rymon, R.: Search through systematic set enumeration. In: Proc. 3rd Int'l Conf. on PKRR

Data Privacy against Composition Attack*

Muzammil M. Baig[1], Jiuyong Li[1], Jixue Liu[1], Xiaofeng Ding[1], and Hua Wang[2]

[1] School of Computer and Information Science,
University of South Australia, Mawson Lakes, SA. 5095 Australia
{muzammil.baig,jiuyong.li,jixue.liu,xiaofeng.ding}@unisa.edu.au
[2] Department of Maths & Computing,
University of Southern Queensland, Toowoomba, Queensland, 4350 Australia
wang@usq.edu.au

Abstract. Data anonymization has become a major technique in privacy preserving data publishing. Many methods have been proposed to anonymize one dataset and a series of datasets of a data holder. However, no method has been proposed for the anonymization scenario of multiple independent data publishing. A data holder publishes a dataset, which contains overlapping population with other datasets published by other independent data holders. No existing methods are able to protect privacy in such multiple independent data publishing. In this paper we propose a new generalization principle (ρ, α)-anonymization that effectively overcomes the privacy concerns for multiple independent data publishing. We also develop an effective algorithm to achieve the (ρ, α)-anonymization. We experimentally show that the proposed algorithm anonymizes data to satisfy the privacy requirement and preserves high quality data utility.

Keywords: Data anonymity, privacy, composition attack.

1 Introduction

Existing privacy preserving data publishing techniques focus on one-time publication [11,8,4] and multiple views of the same data [17]; recently address the scenario of re-publication by single data holder [14,15]. Specifically, privacy preserving data republication is restricted to single data holder, and does not support overlapping population by multiple publishers. The seminal work [2] firstly identify the breach of privacy of existing anonymization methods in multiple independent data publishing, called 'composition attack'. However, the solution of [2] supports only *interactive setting* (where only data statistics and/or query results are published), and is inapplicable for *non-interactive* setting (where the data needs to be published after anonymization). Independent data publishing of overlapping subset by multiple publishers in non-interactive setting remains an open problem.

To illustrate the problem, consider Table 1(a) of Hospital-1. *Identifier attribute(s)* can directly identify individuals, such as Name, SSN etc. They should be removed in a published dataset. *Quasi identifier (QIDs) attributes* could indirectly lead to the identification of individuals in a dataset, such as Age, Zipcode and Sex etc. They are normally generalized so that no individuals are identifiable in a generalized table.

* This research has been supported by ARC Discovery grants DP0774450 and DP110103142.

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 320–334, 2012.
© Springer-Verlag Berlin Heidelberg 2012

Table 1. Patient data and its generalization at Hospital-1

(a) Original data Q_1

Identifier Attribute	*Quasi-Identifiers*		*Sensitive Attribute*
Bob	15	male	*B*
Hudson	45	male	*H*
Robi	40	female	*G*
David	20	male	*B*
Khan	25	male	*C*
Victor	50	male	*H*

(b) Generalized Q_1^*

Group ID	**Age**	**Sex**	**Disease**
1	15 – 25	male	*B*
			B
			C
2	40 – 50	*	*G*
			H
			H

The *Sensitive attribute* contains the private information about the individuals that needs to be protected such as Disease, Income etc. A *generalized* table is considered privacy preserving, if it satisfies a privacy constraint, such as k-anonymity [11] or ℓ-diversity [8]. For example Table 1(b) is 3-anonymous and 2-diverse version of Table 1(a). In other words, 3-anonymity means that values in the QIDs have at least 3 identical copies. So one could not be distinguished from other 2 records. 2-diversity means that each of such a group has at least 2 distinct values in the sensitive attribute. So, the sensitive value of each individual could not be guessed with a high confidence.

1.1 Problem Description and Motivation

Consider the patient overlapping scenario of three hospitals in figure of Table 2(c); David from Hospital-1 and Eliza from Hospital-2 were referred to Hospital-3 so the data of Hospital-3 also include the records of David and Eliza. For simplicity, we omit the overlapping scenario between Hospital-1 and Hospital-2; although our solution provides the privacy protection in any overlapping scenario.

Hospital-3 anonymized its dataset and release it as Table 3(b). Assume that an adversary knows David's QIDs (20 years old male), and the fact that David has visited both Hospital-1 and Hospital-3. The adversary would find the records of David in both hospitals since only one record matches David's QIDs and has the same disease in Table 1(b) and 3(b) respectively i.e. $\{B\}$. Therefore, David is identified in the anonymized datasets of both hospitals. The understanding remains the same for Eliza where adversary can get her disease $\{R\}$ using her QIDs in Table 2(b) and 3(b).

A patient may visit more than one hospitals of his/her area and we assume that hospitals visit information is available in public domain i.e. adversary knows about the hospitals visited by a patient. Moreover, each hospital also knows about other hospitals where it can have overlapping patients. Both are realistic assumptions. Firstly, an adversary is a person that is close to the patient (i.e. a friend, a colleague or a neighbor) and it is reasonable to believe that s/he is aware of the hospitals visited by the patient. Secondly, hospitals visit information is also part of a patient medical record so each hospital also knows about other hospitals where it can have overlapping patients. Although, each hospital knows about the overlapping with other hospitals but each hospital does not (due to internal privacy policies) or cannot (due to legal restrictions) share its original data with another organization.

The problem of overlapping data publication is not resolvable by the methods of sequential data publication, such as m-invariance [15]. m-invariance deals with two

Table 2. Patient data and its generalization at the Hospital-2

(a) Original data Q_2

Name	Age	Sex	Disease
Eliza	40	female	*R*
Arthur	30	male	*M*
Paul	20	male	*M*
Noreen	45	female	*S*
Mathew	15	male	*Q*
Panama	35	female	*T*

(b) Generalized Q_2^*

Age	Sex	Disease
15 – 30	male	*M*
		Q
		M
35 – 45	female	*R*
		S
		T

(c) All overlapping patients

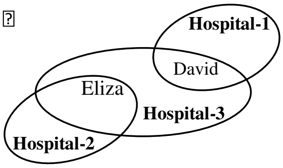


Table 3. Patient data and its generalization at the Hospital-3

(a) Original data P

Name	Age	Sex	Disease
David	20	male	*B*
Anthony	35	male	*C*
Rick	30	male	*C*
Stewart	30	male	*L*
George	28	male	*B*
Smith	38	male	*W*
Eliza	40	female	*R*

(b) Generalized P^*

Age	Sex	Disease
20 – 30	male	*B*
		C
		C
30 – 40	*	*L*
		B
		W
		R

(c) P^* with α-overlap

Age	Sex	Disease
15 – 35	male	*B*
		C
		C
		L
28 – 45 (1)	*	*B*
		W
		R
		S

overlapping data publications of the same data holder by employing the same publication scheme. In multiple publication scenario datasets are more than two, released from different data holders, and mostly anonymized by different publication schemes. Our problem is different from sequential publication and more details are in Section 4.2.

In this paper, our proposed method (ρ, α)-anonymization (details later in Section 4) leads to the publication of Table 3(c) at Hospital-3. Now an adversary has at most 50% chance (in this simple example) to guess the sensitive value of any overlapping individual. Let us reconsider the adversary who has the precise QIDs detail of David and attempts to infer the disease of David from Tables 1(b) and 3(c). S/he can locate that the tuple of David must have been generalized in the first QID groups of Tables 1(b) and 3(c), respectively. These groups encompass the 2 common sensitive values i.e. $\{B,C\}$. Therefore adversary cannot get any specific disease that David has contracted. In case of Eliza, there are also two candidate diseases i.e. $\{R,S\}$. There is one 'counterfeited' tuple (shown in parentheses) in the QID group of Eliza because there was no $\{S\}$ disease in Table 3(a) (details later in Section 5).

1.2 Contributions

This paper presents the first model to prevent the composition attack in non-interactive data publishing setting by combining sampling and generalization. Our solution integrates two novel concepts: (ρ, α)-*anonymization* and *composition-based generalization*. The former is a new anonymization mechanism, which overcomes the drawbacks of generalization by combining it with sampling and provide privacy protection for composition attack. The latter is a technique that facilitates the enforcement of privacy, in the presence of overlapping population.

Secondly, we design an efficient algorithm to compute anonymous datasets that conforms to (ρ, α)-anonymization. Our algorithm aims to maximize the utility of the released data, by minimizing **(i)** the number of counterfeited tuples, and **(ii)** the amount of generalization on the QIDs. Furthermore, the algorithm is versatile, namely, it enables a data holder to produce an anonymized release, by consulting any number of already published anonymous releases of other data holders.

2 Fundamental Definitions

Let P be a dataset maintained by a data holder. There are n other published datasets $Q_1^*, Q_2^*, \ldots, Q_n^*$ which have overlapping population with P. Each published dataset Q_i^* ($i \in$ 1,2,3,...,n) is independently anonymized from its original dataset Q_i.

We classify the columns of P and Q_i ($i \in$ 1,2,3,...,n) into three types (already explained in Section 1): **(i)** an identifier attribute A^{id}, which is the primary key of P, **(ii)** d quasi-identifier (QIDs) attributes A_1^{qi}, A_2^{qi}, ..., A_d^{qi}, and **(iii)** a sensitive attribute A^s. The QIDs can be either numerical or categorical. For each tuple $t_p \in P$, $t_p[A]$ denotes its value on attribute A.

Definition 1 (Generalized QID group / Equivalence class). *For an anonymous dataset P^*, a generalized QID group is subset of the tuples in P. Each generalized QID group is assigned an unique ID A^g. All tuples in P^* with the same A^g have the identical values in QID attribute.*

For a tuple $t_p^* \in P^*$; the $t_p^*.QI$ denotes such generalized QID group which has t_p^* in P^*. We refer to $t_p^*.QI$ as the 'generalized QID hosting group' of the t_p^* in P^*. Next, we introduce an important notation OL.

Definition 2 (Overlapping Set). *For dataset P and each already published independent anonymous dataset $Q_i^*(i \in 1, 2, 3 \ldots, n)$, the overlapping set* (OL) *contains all those tuples in P such that:*

$$\mathrm{OL} = \bigcup_{i=1}^{n} (Q_i^* \cap P) \;\; (i \in 1, 2, 3 \ldots, n)$$

Each tuple $t \in$ OL is an intersection (to be explained) of two corresponding tuples $t_i^* \in Q_i^*$ and $t_p \in P$; who satisfy the following properties:

1. $t_i^*[A^s] = t_p[A^s]$*; both tuples have same sensitive value and*
2. $t_i^*[A_j^{qi}] \cap t_p[A_j^{qi}] \neq \emptyset, (1 \leq j \leq d)$*; t_i^* and t_p have overlapping value interval in j-th QID attribute.*

Note that none of the data holder shares its original data with another data holder. Rather, before anonymizing its original data, the data holder of P gets the publicly available anonymous datasets of other data holders, i.e. Q_1^*,Q_2^*, ..., Q_n^*, and computes the overlapping set (OL) using Definition 2. After that the data holder of P applies our anonymization technique (described in detail in Section 4).

For numeric QIDs, the intersection in Definition 2 returns the overlapping value. For example, the intersection of *age* QID value 15–25 in Q_i^* and 20 in P returns 15–25 $\cap$ 20 $=$ 20. For categorical QIDs, the intersection returns the value of the closest common generalization of two values. For example, intersection of values *'male'* $\cap$ *'female'* $=$ '$\emptyset$'. If one value is the generalization of another value, the intersection returns the more specific value. For example, the intersection of '$*$'$\cap$ *'male'* $=$ '$male$'. Here '$*$' corresponds to most generalized QID value in any generalization hierarchy. In sex QID generalization hierarchy, '$*$' presents both $male$ and $female$.

3 Cases of Privacy Breach in Composition Attack

3.1 Pros and Cons of Sampling in Composition Attack

An apparent way to combat the composition attack is sampling, i.e. only publish a portion of data. After a dataset is sampled, an adversary does not know if the record is in the published dataset or not. However, sampling only reduces the chance of finding overlapping tuples, but does not reduce the confidence of an adversary for inferring the sensitive information once overlapping tuples are found.

Example 1. Let us assume that the true match is caused by the same person visiting two hospitals, and that a false match is caused by two unrelated patients who happened to have the same QIDs and disease in two datasets. Let the sample rate be 50%. The probability of a true match is 25% when a patient have visited two hospitals. The chance of two unrelated patients to have the same QIDs (false match) depends on the data distributions of two datasets. For a simple illustration, let us assume that the chance is 50%. Assume that there are 5 sensitive values and each has the same chance to associate with QIDs. The chance for two QIDs matched tuples to have the same disease is only 4% and this reduces the probability of a false match down to 2%; which is much less than the 25% probability of true match. Therefore, an adversary has a reason to be confident about true match.

3.2 Pros and Cons of Generalization in Composition Attack

Let us assume that two or more data holders achieve ℓ-diversity [8] in the overlapping set (OL); such that overlapping equivalence classes have at least ℓ overlapping patients common that are suffering from distinct diseases (although it is not trivial to achieve this, and we discuss it in the following section). Intuitively, adversary only learns that an overlapping victim suffering from one of ℓ possible diseases. However, the privacy is possibly compromised for non-overlapping victim(s).

Example 2. In published datasets P^* and Q_i^*, there are QID groups as $P^* = \{$31–35, male, $(A,B,C)\}$ and $Q_i^* = \{$31–35, male, $(A,B)\}$. The adversary has only 50% chance of knowing if two overlapping victims who visited both P^* and Q_i^* suffer from disease $\{A\}$ (or $\{B\}$). However, the adversary has a chance to learn the sensitive information of a victim that is not in the overlapping set (OL). For example, the adversary knows a victim (male, 31) who only visited P^*. Based on the above data publication, the adversary knows that the victim (male, 31) suffers from disease $\{C\}$.

Data publication by generalization also suffers the minimality attack [13]. An adversary can use the knowledge of an anonymization algorithm to infer the sensitive information of individuals. The same attack applies to multiple data releases.

Example 3. Assume that an algorithm follows the following procedure. If the overlapping set (OL) satisfies ℓ-diversity, publish the data section. Otherwise, generalize the data section with the adjacent tuples to make the overlapping set (OL) satisfy ℓ-diversity. If not possible, suppress tuples to make the overlapping set (OL) empty. Based on the principle, P^* is published using the information of already published $Q_i^* = \{$31–35, male, (*A*,*B*)$\}$ and $P^* = \{$35–40, male, (*A*,*C*)$\}$. The adversary knows that victim (male, 33) visited both Q_i^* and P^*. Based on the published datasets, he does not know if the victim suffers from disease $\{A\}$ or $\{B\}$. The adversary knows that the victim's record has been suppressed from the subsequent dataset P^*, but this information does not help her/him to figure out the true sensitive information of the victim either. However, s/he knows the generalization algorithm as well. S/he reasons the sensitive value of victim as disease $\{A\}$ as the following. If the victim suffers from disease $\{B\}$, the subsequent published dataset P^* should be as $P^* = \{$33–40, male, (*A*,*B*,*C*)$\}$ to maintain the 2-diversity in the overlapping set (OL). The record of the victim is suppressed from P^* because the victim does not suffer from disease $\{B\}$ and there is no possibility to generalize the data to satisfy 2-diversity in the overlapping dataset (OL). Therefore, the victim suffers disease $\{A\}$ for sure.

4 (ρ, α)-Anonymization Model

ρ-sampling and α-overlapping, in short (ρ, α)-anonymization, model consists of two steps anonymization, as detailed in the following.

Definition 3 **(ρ-Sampling).** *Given a sampling probability $\rho \leq 1$, each tuple $t \in P$ is sampled with the probability of ρ without replacement, i.e. whether a tuple is included in sampled dataset P^ρ for subsequent publication is decided by tossing a coin with head probability ρ. Only if the coin heads, a tuple is included in P^ρ.*

Sampling is already a routine practice in data publishing [12], because data publishers hold gigantic data and only a subset is publicly released. As shown in previous section, an adversary infers the sensitive values of individuals in overlapping and non-overlapping datasets with different confidences. The sampling is necessary for privacy protection of non-overlapping tuples in multiple independent data releases since it reduces the confidence of locatability of an adversary. Later, in Section 4.1, we discuss in detail how sensitive value inference of a non-overlapping tuple is bounded by sampling. Next we preserve the privacy of overlapping tuple(s).

Definition 4 **(α-overlap).** *Independently published anonymous dataset P^* (formed from P^ρ) satisfies α-overlap, if for any tuple $t \in \text{OL}$; its QID group in P^* contains at least α ($\alpha \geq 2$) uniformly distributed distinct sensitive values with Q_i^*.*

The overlapping set (OL) is computed by utilizing publicly available anonymous releases of other publishers using Definition 2. The rationale of α-overlap is that, if a tuple t is published by more than one publishers then all its generalized QID hosting groups must contain α common sensitive values in a way such that its sensitive values in QID hosting groups forms uniform distribution (i.e. equal number) for α common sensitive values in overlapping set (OL). The uniform distribution in overlapping set (OL)

makes an adversary's confidence equally split over α sensitive values. The distributions of the sensitive values in P^* and Q_i^* can be quite different. The uniform distribution is a good trade-off between diverse distributions.

4.1 Privacy Analysis of (ρ, α)-Anonymization

In this section we analyze the privacy of overlapping and non-overlapping tuples in (ρ, α)-anonymization. We start with the privacy of non-overlapping tuples.

Observation 1. *If dataset P^* satisfies (ρ, α)-anonymization, then the confidence of an adversary to derive the true sensitive value of any non-overlapping tuple t from P^* is bound by sampling probability ρ.*

Example 4. Reconsider the scenario of Example 2 with additional assumption that P^* is sampled with 50% probability. Now, adversary has maximum ρ chance that the record <male, 31–35, *C*> is the one s/he is looking for and there is no other source of information to reinforce this.

Next we reason about the privacy of overlapping tuples in (ρ, α)-anonymization.

Observation 2. *If dataset P^* satisfies (ρ, α)-anonymization with all already published anonymous datasets, then the confidence of an adversary to derive the sensitive value of any overlapping tuple $t \in \mathrm{OL}$ through the composition attack is bound by $\lceil \frac{1}{\alpha} \rceil$; where $\lceil\ \rceil$ is ceiling operator.*

An ideal situation is that the confidence of guessing a sensitive value by an adversary from an anonymous dataset is similar to the distribution of sensitive value in original data, like in t-closeness [7]. However, in the composition attack, we deal with more than one datasets which may have different distributions for sensitive values. We do not have a "standard" distribution to close to. Further, the overlapping set (OL) is a small proportion of a dataset, and may not represent the distribution of the global dataset. Uniform distribution for α common sensitive values is a good trade off. Any latter data holder, (who is at the risk of composition attack i.e. Hospital-3 in our case) can simply set α to a sufficiently larger value, for every overlapping tuple $t \in \mathrm{OL}$, to achieve the required extent of privacy preservation.

Example 5. Reconsider the scenario of Example 3, where the P^* will be $P^* = \{31\text{–}40$ (1), male, (*A*,*B*,*C*)$\}$ to maintain the 2-overlap with Q_i^*. Note that, although the adversary learns that a counterfeit exits in QID group of P^*, s/he still cannot narrow down the possible diseases of overlapping victim (male, 33). In fact, to the adversary, there is a 50% chance that either $\{A\}$ or $\{B\}$ would be the counterfeit.

4.2 m-Invariance: Similar Model But Not Good in This Scenario

m-invariance [15] model is a very typical model in serial data publication. It has certain strengths for privacy protection in multiple data publications, but it has its limitations in our problem. First, it needs a sampling process for m-invariance model in our scenario too. Second, m-invariance model requires every tuple in an overlapping dataset

Table 4. Major symbols used in different phases of composition based generalization

P	Input dataset to be anonymized	$Q^*[\,]$	Already published overlapping datasets
ρ	Input sampling parameter for dataset P	β	Input parameter, a trade-off for efficiency and quality
k	Input parameter for k-anonymity	ℓ	Input parameter for ℓ-diversity
α⍰	Minimum uniformly distributed distinct sensitive values to be placed in overlapping set OL		

to persistently associate with the same set of sensitive values, called *signature* in [15]. In our scenario, the number of counterfeit or suppressed tuples must be large to satisfy the *persistent consistency* as required in m-invariance [15]. m-invariance requires every overlapping tuple in P^* to associate with m number of same sensitive values, (*signature* as defined in [15]), as the corresponding tuple in Q_i^*. In other words, m-invariance requires all sensitive values (both overlapping and non-overlapping) in QID groups of datasets P^* and Q_i^* with overlapping QID values be the same; whereas we only need to handle overlapping QID groups to combat composition attack.

Example 6. Let the sensitive values of the QID groups in $Q_1^* = \{A,B\}$, $Q_2^* = \{A,C\}$, $Q_3^* = \{A,D\}$ and the available sensitive values in $P^\rho = \{A,B,C\}$ (let $P^\rho = P$ with $\rho =$ 50%). Now to meet 2-invariance requirement we need 3 QID groups with counterfeit tuples $\{\emptyset\}$, $\{A\}$ and $\{A,D\}$ respectively. In contrast, we require one counterfeit tuple, i.e. $\{D\}$, to meet (50%, 2)-anonymization. Intuitively, m-invariance [15] principle is too strong in our scenario.

5 Composition Based Generalization

5.1 Phases

We use the running example to demonstrate the different phases of composition based anonymization to achieves (ρ,α)-anonymization; where $\rho = 50\%$, $k = 4$, $\ell = 3$, $\alpha = 2$, Q_1^*, Q_2^* and P are Tables 1(b), 2(b) and 3(a) respectively. Given already published tables Q_1^* and Q_2^* available to data holder of P, we show how to compute the anonymized version P^* from P. We perform the computation in following five phases: *sampling, division, balancing, assignment* and *generalize*. The explanation of the major symbols used in different phases of composition based generalization is shown in Table 4.

Sampling. Firstly, we apply the sampling on P with input sampling probability ρ to obtain P^ρ. Each tuple of P is independently sampled. In our example, we assume sampling probability $\rho = 0.5$ and sampling function $f_\rho(t)$ returns the tuple (i.e. $f_\rho(t) = t$) if $f_\rho = 1$ and $f_x(t) = \emptyset$ if $f_\rho = 0$. In our case the probability of getting {0,1} is 0.5. In our example, we assume that for all the tuples of P, $f_x(t) = t$, i.e. $P^\rho = P$.

Division. In this phase we partition the sampled P^ρ into two disjoint sets i.e. *overlap tuples* $S_\cap = Q_i^* \cap P$ $(i \in 1, 2)$; computed as per Definition 2 and *non-overlap tuples*

$S_- = P^\rho - S_\cap$. In case of our example the tuples with sensitive values $\{B,R\}$ and $\{C,C,L,B,W\}$ are included in $S_\cap$ and S_- respectively. For each tuple $t_\cap \in S_\cap$, we define its 'possible sensitive values' as the set of distinct sensitive values in the corresponding generalized QID hosting group in already published Q_i^*. In the running example the tuples, with sensitive value $\{B\}$ in $S_\cap$, has possible sensitive values as $\{B,C\}$; i.e. the distinct sensitive values in QID group-1 of Table 1(b). Whereas the set of possible sensitive values for $\{R\}$ is $\{S,R,T\}$; i.e. the distinct sensitive values in QID Group-2 of Table 2(b).

In the end of division phase, we simply divide $S_\cap$ into several QID groups, on the basis of their possible sensitive values. In our running example, we have two QID groups i.e. $GRP_1(B,C)$ and $GRP_2(S,R,T)$.

Balancing. We say that a QID group GRP_i $(i \geq 1)$ is *balanced*, if it contains at least α tuples; having such distinct sensitive values that these α tuples along with their corresponding QID group(s) in already published overlapping dataset(s) comply with α-overlap principle (Definition 4). For example, the QID group GRP_1 will become balance with corresponding overlapping QID group-1 of Table 1(b) if we include two tuples (from S_-) having sensitive value $\{C\}$ $(\alpha = 2)$. The objective of this phase is to balance all QID groups.

Continuing our example, we cannot balance QID group GRP_2 with corresponding QID group-2 in Table 2(b) because to balance GRP_2 we need at least one tuple having either of sensitive values $\{S,T\}$ in S_-. As there is no such sensitive values in S_- so we add one counterfeit sensitive value (either of $\{S,T\}$) in the QID group GRP_2 to make it balance with corresponding QID group-2 in Table 2(b) . We add counterfeit sensitive value, in unbalanced QID group GRP_2, instead of suppressing the overlapping tuple because suppression can still breach the privacy of overlapping tuple, as shown in Example 3. A non-desiring solution can be to suppress all the tuples of P^ρ with the same sensitive values as of corresponding QID group in Q_i^*.

Assignment. We assign remaining tuples of S_- (if any) in two steps. First, we include the tuples in existing QID group(s) to comply with the generalization principle(s) (k-anonymity [11], ℓ-diversity [8], t-closeness [7] etc.). Second, if necessary, new QID group(s) may be created for remaining tuples. A QID group is called *complete* if it complies with all underlaying generalization principles. The purpose of this phase is to make all QID groups complete. In running example, S_- has three tuples having sensitive values $\{L,B,W\}$. We have assumed that $k = 4$ and $\ell = 3$ as generalization principles. The tuple having sensitive value {L} is assigned to GRP_1 and remaining two tuples having sensitive values {B,W} are assigned to GRP_2 to make both groups complete.

The crucial part is the selection of a tuple t_- from S_- and its assignment to a QID group. We select first β tuples of S_- and search for the *optimal tuple* which requires least anonymization of QIDs. β is an input parameter that restricts the search of the optimal tuple within the first β tuples of S_-. The incorporation of β improves the performance of assignment process (as shown in experiments in Section 6) because instead of traversing all the tuples of S_-, only β tuples are searched for optimal tuple. Note that

Algorithm 1. *overlapAnonymize*(P,$Q^*[\,]$,ρ,β,α,k,ℓ)

```
1: $P^\rho$                                          ▷ sample each tuple of $P$ with $\rho$ probability to get $P^\rho$
2: $S_\cap = Q^*[\,] \cap P^\rho$                    ▷ get overlap tuples (Definition 2)
3: $S_- = P^\rho - S_\cap$                           ▷ get non-overlap tuples
4: Divide $S_\cap$ into $GRP[\,]$ QID groups          ▷ *Division* phase
5: $Sort(S_-)$                                       ▷ sort all $S_-$ tuples on the basis of QIDs
6: while $|S_-| \neq 0$ do                           ▷ continue till all tuples are assigned
7:     for $i \leftarrow 1,\ \beta$ do               ▷ to access first $\beta$ tuples of $S_-$
8:         $t_- = S_-[i]$
9:         for $j \leftarrow 1,\ |GRP|$ do           ▷ to access all groups
10:            if $GRP[j]$ is complete with $k$, $\ell$ and $\alpha$ then exclude $GRP[j]$ from $GRP[\,]$
11:            else if $t_-$ is optimal to $GRP[j]$ then assign $t_-$ to $GRP[j]$
12:            if $|GRP| == 0$ then $i = \beta; j = |GRP|$ ▷ set to break the loops of lines 7 and 9
13:        end for
14:    end for
15:    if $|GRP| == 0\ AND\ |S_-| \geq k$ then create new $GRP[0]$    ▷ all groups become
       *complete* but still there are more than $k$ (from $k$-anonymity) unassigned tuples
16:    else if $|GRP| == 0\ AND\ |S_-| < k$ then assign all remaining tuples to recently com-
       pleted group
17:    else if $|GRP|\ ! = 0\ AND\ |S_-| == 0$ then *complete* all remaining groups by adding
       counterfeit tuples.
18: end while
```

the anonymization of optimal tuple depends on the specific generalization principle(s) to be employed. Within β tuples, we calculate *Distortion* [6] caused by every tuple t_- $\in S_-$ to each QID group and assigns such t_- (also referred optimal tuple) to the QID group which has minimum distortion with t_-; as long as α-overlap (Definition 4) holds. Due to space constraint, we are omitting the calculation details of distortion between QID group and a tuple and reader is referred to the original paper [6] for further details. Algorithm 1. formally presents the assignment strategy.

In our running example we assume $k = 4$ and $\ell = 3$ and there are two QID groups; GRP_1 contains 3 tuples and GRP_2 has two tuples. We have three tuples in S_- with sensitive values $\{L,B,W\}$, as per their order in P^ρ. Due to the sorting of the S_- (Algorithm 1 line 5) on the basis of QIDs; the sorted tuples have the sensitive values as $\{B,L,W\}$. In first iteration we assign the tuple with sensitive value {L} to GRP_1 to make it complete i.e now GRP_1 has four tuples ($k = 4$), three distinct sensitive values ($\ell = 3$) and two overlap values ($\alpha = 2$). So, we exclude the GRP_1 from all subsequent iterations (Algorithm 1 line 10). Importantly, we cannot assign the tuple $\{B\}$ to GRP_1 instead of tuple $\{L\}$ (although the tuple $\{B\}$ requires less generalization for GRP_1) because after the assignment of tuple $\{B\}$ to GRP_1, the GRP_1 will not comply with α-overlap condition (Definition 4). Now, we are left with two tuples with sensitive values $\{B,W\}$ (i.e. $|S_-| \neq 0$) and GRP_2 is incomplete (as $k = 2$ instead of 4); so next two iterations assign the $\{B,W\}$ tuples to GRP_2 (Algorithm 1 line 11) and assignment algorithm finishes by breaking assignment loop (Algorithm 1 line 6).

Table 5. Attribute domain size

Attribute	Age	Sex	Education	Marital Status	Birth Place	Occupation
Domain Size	91	2	17	7	50	50

Generalize. We can have two types of QID groups, i.e. *overlap QID groups* (created during division phase) and *non-overlap QID groups* (created in assignment phase). The generalization of non-overlap QID group is trivial; we get the minimum QID range that covers all the QID values and replace the original QID values with this range.

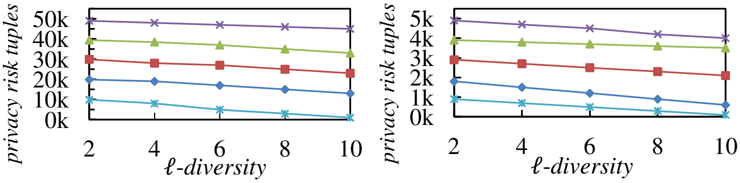
In case of overlap QID group, we get the minimum QID range that **(i)** covers all QID values in current QID group (similarly like non-overlap group) **(ii)** as well as the QID generalization range in corresponding overlap QID group of already published dataset. In our running example, the generalization range of the age in overlap QID group GRP_1 will be (15 – 35). We cannot put (20 – 35) as generalization range for age in GRP_1 because (20 – 35) only covers age QID values in GRP_1; whereas the generalization range of age in corresponding overlap group of already published Q_1^* is (15 – 25). So the minimum range for age in GRP_1 is (15 – 35) instead of (20 – 35). There is no need to generalize sex in GRP_1 because without any generalization the aforementioned both conditions of overlap group are met. In the same way, the generalization range for age in GRP_2 is (28 – 45) instead of (28 – 40). The Table 3(c) is the final outcome after generalization phase.

The complexity of the Algorithm 1 mainly depends on the computation of tuples in $S_\cap$ (Algorithm 1, line 2), sorting of S_- tuples (Algorithm 1, line 5) and searching the optimal tuple (Algorithm 1, line 11). We assume that $|P| \approx |Q_i^*|$, so major computation overhead lies on the computation of $S_\cap$ i.e. $|P| * |P|$ steps. Intuitively, the complexity of $O(m^2)$ where $m = |P|$. The more optimization of anonymization algorithm is future work we plan to pursue.

6 Experiments

All the experiments are performed on a machine running a 2.4 Ghz CPU with 3 Gigabyte memory. We deploy two real repositories BIR and OCC from United States census data downloadable from http://ipums.org. Both contain 300k and 45k tuples respectively. *BIR* includes four QID attributes, age, $gender$, $education$, $marital\ status$, and a sensitive attribute $birth\ place$. Whereas OCC contains the same QID attributes, but with different sensitive attribute $occupation$. All columns are discrete with domains size given in Table 5.

We create four disjoint sub-datasets Q_i^{bir} (Q_i^{occ}) ($n \in 1, \ldots, 4$) from $BIR(OCC)$. It suffices to clarify the generation and generalization of Q_i^{bir}, since the same method is used for Q_i^{occ}. Each sub-dataset Q_i^{bir} is of 50k tuples. Next, we form the dataset P_{bir}, also having 50k tuples. The remaining 50k tuples initiates a pool $^{O}_{bir}$. The dataset P_{bir} will be anonymized using (ρ, α)-anonymization. For OCC each sub-dataset Q_i^{occ} and P_{occ} contains 8k tuples and remaining 5k tuples goes in pool $^{O}_{occ}$. In all experiments we set sampling probability $\rho = 0.50$.



(a) P^*_{bir} (b) P^*_{occ}

Fig. 1. Successful composition attack vs. the overlap volume u

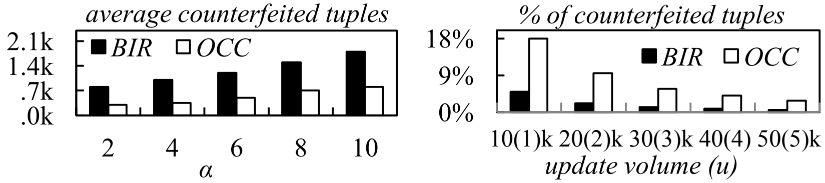
We run four sets of experiments, involving 4 sub-datasets with P_{bir}. In each set of experiment, the publication of n sub-datasets is straightforward, i.e we randomly select $u * n$ tuples from $^{O}_{bir}$ and insert separate u tuples in each Q_i^{bir}, subsequently anonymous Q_i^{bir*} (having 50k + u tuples) is created that satisfies some generalization principle(s). Here u is a parameter, called *overlap volume*, controlling the overlap rate between n sub-datasets and P_{bir}. For the P_{bir}, the generalized P^*_{bir} is obtained by, first, inserting the same $u * n$ tuples in P_{bir}; i.e. P_{bir} has separate u overlap tuples with each Q_i^{bir*}. Consequently P_{bir}, having 50k + $(u*n)$ tuples, is anonymized using (ρ, α)-anonymization that utilizes other four anonymous sub-datasets Q_1^{bir*}, Q_2^{bir*}, $\ldots$, Q_4^{bir*}. We repeat this process by increasing the u from 10k to 50k (i.e. each set of experiment includes 5 iterations on BIR). In case of OCC, we increase the overlap volume u from 1k to 5k tuples in each set of experiment, so total iterations in each set of the experiment of the OCC are also 5.

6.1 Failure of Conventional Generalization Schemes

In the first set of experiments, we aim at establishing the conjecture that the existing generalization principles may lead to severe privacy disclosure in independent data publishing. This finding was also observed in [2]. We adopt the algorithm in [5] to compute ℓ-diversity [8] as the representative generalization principle, since it is widely adopted and offers stronger privacy than k-anonymity [11]. In Fig. 1(a), we plot the number of privacy risk tuples (as explained in Section 1.1) in P^*_{bir} as a function of u, as this parameter changes from 10k to 50k in $^{O}_{bir}$. Regardless of u and ℓ, there are nearly 90% overlap tuples whose privacy is not preserved at all in P^*_{bir}. We repeat the experiments on sub-datasets Q_i^{occ} and P_{occ}. The results are illustrated in Fig. 1(b), confirming the same observations.

6.2 (ρ, α)-Anonymization Evaluation

We have $n = 4$ already published sub-datasets of BIR and OCC i.e. Q_1^{x*}, Q_2^{x*}, $\ldots$, Q_4^{x*} ($x = BIR$ or OCC). We invoke the (ρ, α)-anonymization (Section 4) on P^x to compute the generalized version P^*_x for α-overlap publication. The computation of P^*_x utilizes already published four anonymous datasets to identify overlapping set (OL) using Definition 2. The P^*_x is characterized by two input parameters, i.e. overlap-volume u and overlap diversity α.



(a) *update volume* $u = 10(1)$k

(b) $\alpha = 6$

Fig. 2. Average and Percentage of counterfeiters in P^*_{bir} (P^*_{occ})

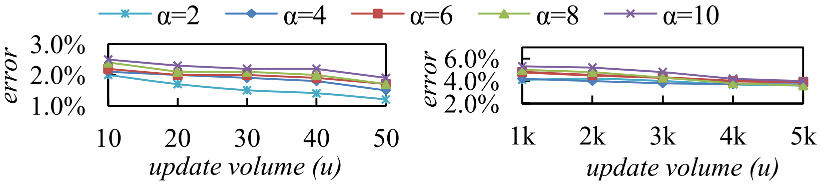
Number of Counterfeited Tuples. We start by demonstrating that only a small number of counterfeited tuples are needed to enforce (ρ, α)-overlap. In Fig. 2(a), we set u = 10(1)k but vary the α from 2 to 10 and measure the average counterfeited tuples in P^*_{bir} (P^*_{occ}). The average number of counterfeited tuples increase along with α because higher α requires more distinct sensitive values in each overlap QID group for balancing; failure of this causes insertion of more counterfeited tuples in QID group(s).

Next, we focus on the percentage of counterfeiters with overlap-volume u. We get the percentage of counterfeited tuples in P^*_{bir} (P^*_{occ}) for all sub-datasets of BIR (OCC). Fixing $\alpha = 6$, Fig. 2(b) shows the percentage of counterfeited tuples for both BIR and OCC as a function of u. The percentage decreases as u increases, such that (ρ, α)-overlap can utilize more overlapping tuples. This is expected, because for a fix value of α as u increases, more overlap QID groups are more likely to have same set of possible sensitive values which can accommodate larger overlap volume. Intuitively, causing less counterfeited tuples while balancing.

Utility of the Published Data. In the following set of experiments, we will use P^*_x (where $x = BIR$ or OCC) to answer queries about the original sub-dataset P_x. We concentrate on aggregate queries, since they are the basic operation for numerous mining tasks (e.g., decision tree learning, association rule mining etc.). Specifically, each query has the form:

SELECT COUNT (*) FROM P^*_x WHERE $pred\{$ $t^*_x[A^{qi}_1]$ AND … AND $t^*_x[A^{qi}_4]$ AND $t^*_x[A^s]$ $\}$

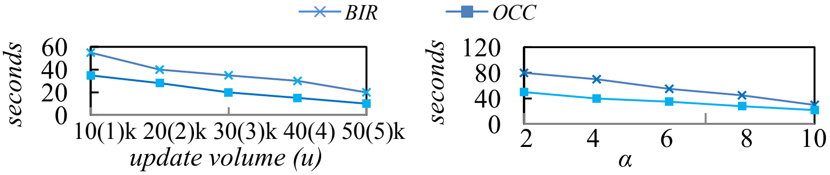
The P^*_x is the sub-dataset generalized using (ρ, α)-overlap, $t^*_x[A^{qi}_1], \ldots, t^*_x[A^{qi}_4]$ denote the four QID attributes in P^*_x, and $t^*_x[A^s]$ is the sensitive attribute $birth\ place$ ($occupation$). For each attribute A, the condition $pred(A)$ has the form $|A|.\delta$, where $|A|$ is the domain size of A (see Table 5), and δ is a query parameter called selection range. A larger result is returned with higher δ. Our workload consists of 1000 queries with same P^*_x and $t^*_x[A^s]$. Given a query, we obtain its actual result R_{act} from original overlap sub-dataset P^x, and compute an estimated answer R_{est} from its (ρ, α)-overlap generalized version P^*_x. The relative error of a query equals $|R_{act} - R_{est}|/R_{act}$. We measure the workload error as the median relative error of all the queries. Adopting α = 6, Fig. 3 plots the workload error as a function of update volume u for $P^*_{bir}(P^*_{occ})$ respectively. In all experiments, the error is at most 2.5(5.3)% for α-overlap, indicating high utility of the (ρ, α)-overlap.



(a) P^*_{bir} ($\alpha = 6$)

(b) P^*_{occ} ($\alpha = 6$)

Fig. 3. Query error vs. update volume u



(a) $\alpha = 6$

(b) *update volume* $u = 10(1)$k

Fig. 4. Computation overhead vs. u and α

Computation Overhead. The last experiment evaluates the efficiency of our overlap generalization algorithm. First in Fig. 4(a), we set $\alpha = 6$, and measure the average time of computing a generalized sub-dataset P^*_{bir} (P^*_{occ}) for different u. The cost is more expensive when u is higher, because the algorithm needs to process more tuples of overlap volume in each QID group. Then in Fig. 4(b), we fix u to 10(1)k, and got the cost as a function of α. The overhead decreases as α increases, since a larger α necessitates fewer overlap QID groups, and requires less time in generalization phase.

7 Related Work

Privacy preserving data publishing has mainly focused on taking into account other known releases, such as previous publications by the same data holder (called sequential, serial or incremental releases) [14,15] and multiple views of the same dataset [16,17]. Another line has considered incorporating knowledge from partitioned views of a same dataset to group individuals [16]. The sequential/multiple view release models do not fit because, in this paper, we deal with the case when there are multiple independent publishers but their release is single. A hypothetical discussion of the same problem is in [2] (driving concepts from differential privacy [1]) without the actual implementation and the test results. Although, some recent work has implemented differential privacy in data publishing [10] but it did not address the composition attack. Most relevant works to this paper are [9,3]. But they incorporate the *coordinated model*; where all locations communicate with each other before releasing their data to calculate the privacy risk of overlapping population and subsequently release dataset that is *k-linkable* i.e. each overlapping record is minimum linked to *k* records in each release. The coordinated model also does not comply with our requirement because we are dealing with non-coordinated scenario where each location independently anonymizes its data without having any communication with other location(s). Secondly in coordinated model, all locations anonymize and publish data at the same time but in our case each location can publish its data anytime.

8 Conclusion

Existing single/serial data publishing methods do not support multiple independent data publication by different data holders having overlapping records. This paper has developed (ρ, α)-overlap anonymization model to prevent an adversary from using data releases of different data holders to infer sensitive information of overlapping individuals. We have provided an efficient algorithm for computing anonymized datasets to achieve (ρ, α)-overlap. We experimentally showed that the anonymized data adequately protects privacy and yet supports effective data analysis.

References

1. Dwork, C.: Differential Privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 1–12. Springer, Heidelberg (2006)
2. Ganta, S.R., Kasiviswanathan, S.P., Smith, A.: Composition attacks and auxiliary information in data privacy. In: KDD 2008, pp. 265–273 (2008)
3. Jiang, W., Clifton, C.: A secure distributed framework for achieving k-anonymity. JVLDB 15, 316–333 (2006)
4. Jin, X., Zhang, M., Zhang, N., Das, G.: Versatile publishing for privacy preservation. In: KDD 2010, pp. 353–362 (2010)
5. LeFevre, K., DeWitt, D.J., Ramakrishnan, R.: Mondrian multidimensional k-anonymity. In: ICDE 2006, p. 25 (2006)
6. Li, J., Wong, R.C.-W., Fu, A.W.-C., Pei, J.: Anonymization by local recoding in data with attribute hierarchical taxonomies. TKDE 20, 1181–1194 (2008)
7. Li, N., Li, T., Venkatasubramanian, S.: t-closeness: Privacy beyond k-anonymity and ℓ-diversity. In: ICDE 2007, pp. 106–115 (2007)
8. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: ℓ-diversity: Privacy beyond k-anonymity. In: TKDD (2007)
9. Malin, B.: k-unlinkability: A privacy protection model for distributed data. DKE 64(1), 294–311 (2008)
10. Mohammed, N., Chen, R., Fung, B.C., Yu, P.S.: Differentially private data release for data mining. In: KDD 2011, pp. 493–501. ACM, New York (2011)
11. Sweeney, L.: k-anonymity: a model for protecting privacy. IJUFKS, 557–570 (2002)
12. Tao, Y., Xiao, X., Li, J., Zhang, D.: On anti-corruption privacy preserving publication. In: ICDE 2008, pp. 725–734 (2008)
13. Wong, R.C.-W., Fu, A.W.-C., Wang, K., Pei, J.: Minimality attack in privacy preserving data publishing. In: VLDB 2007, pp. 543–554 (2007)
14. Wong, R.-W., Fu, A.-C., Liu, J., Wang, K., Xu, Y.: Global privacy guarantee in serial data publishing. In: ICDE 2010, pp. 956–959 (2010)
15. Xiao, X., Tao, Y.: m-invariance: towards privacy preserving re-publication of dynamic datasets. In: SIGMOD 2007, pp. 689–700 (2007)
16. Yang, B., Nakagawa, H., Sato, I., Sakuma, J.: Collusion-resistant privacy-preserving data mining. In: KDD 2010, pp. 483–492 (2010)
17. Yao, C., Wang, X.S., Jajodia, S.: Checking for k-anonymity violation by views. In: VLDB 2005, pp. 910–921 (2005)

Protecting Sensitive Relationships against Inference Attacks in Social Networks[*]

Xiangyu Liu and Xiaochun Yang

College of Information Science and Engineering,
Northeastern University, Liaoning, 110819, China
neulxy@gmail.com, yangxc@mail.neu.edu.cn

Abstract. The increasing popularity of social networks in various application domains has raised privacy concerns for the individuals involved. One popular privacy attack is identifying sensitive relationships between individuals. Simply removing all sensitive relationships before releasing the data is insufficient. It is easy for adversaries to reveal sensitive relationships by performing link inferences. Unfortunately, most of previous studies cannot protect privacy against link inference attacks. In this work, we identify two types of link inference attacks, namely, *one-step link inference* attacks and *cascaded link inference* attacks. We develop a general framework for preventing link inference attacks, which adopts a novel lineage tracing mechanism to efficiently cut off the inference paths of sensitive relationships. We also propose algorithms for preventing *one-step link inference* attacks and *cascaded link inference* attacks meanwhile retaining the data utility. Extensive experiments on real datasets show the satisfactory performance of our methods in terms of privacy protection, efficiency and practical utilities.

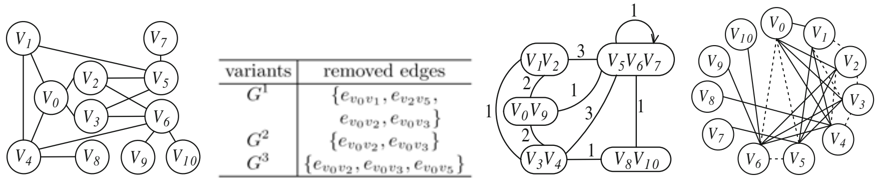
1 Introduction

In recent years, a fast growing popularity in social networks has attracted the interests of researchers from different disciplines. Exploring the properties of social networks has generated interesting knowledge discovery and data mining problems. However, social networks usually contain individuals' sensitive information. Preserving privacy in the release of social network data becomes an important concern.

One fundamental privacy issue in publishing social network data is link reidentification problem. In social network, the main entities are individuals who participate in thousands of interactions with each other. An edge (or link) refers to an interaction between two individuals involved, and there are many different kinds of interactions. In Email network, an edge connecting two people indicates that they communicate through emails. Some interactions or relationships are

[*] The work is partially supported by the National Natural Science Foundation of China(Nos. 61173031, 60973018) and the Fundamental Research Funds for the Central Universities(Nos. N090504004, N100604013).

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 335–350, 2012.
© Springer-Verlag Berlin Heidelberg 2012



variants	removed edges
G^1	$\{e_{v_0v_1}, e_{v_2v_5},$ $e_{v_0v_2}, e_{v_0v_3}\}$
G^2	$\{e_{v_0v_2}, e_{v_0v_3}\}$
G^3	$\{e_{v_0v_2}, e_{v_0v_3}, e_{v_0v_5}\}$

(a) G (b) Variants of G (c) G_a (d) G_1

Fig. 1. Multiple versions of graph G

considered as sensitive due to the involved individuals. For instance, in telecommunication network, *Adam* and *Bob* would like to prevent the disclosure of their close relationship, which can be observed based on the frequent telephone calls between them.

1.1 Motivation

In the process of anonymizing data, sensitive relationships are always removed, i.e., they are not provided in the released data in order to protect individuals' privacy. However, it may be possible to predict or infer some of these relationships through the techniques of *link inference*, which has been widely investigated in recent years. We name inferring sensitive relationships in social network as *link inference attack*. A common sense we have known is that if two individuals have many friends in common, these two individuals are likely to know each other. In Fig.1(a), v_0 has three common neighbors with v_5, and only one common neighbor with v_8. We infer the probability of there existing a link between v_0 and v_5 is larger than that between v_0 and v_8. Link inference techniques could be adopted by adversaries to identify sensitive relationships with high probability. We empirically evaluated the re-identification power of link inference on real social networks, **Email-1** and **LiveJ-1**[10], to demonstrate that such intuition really holds. In our evaluation, for any two vertices, we considered there existed an edge between them if the count of their common neighbors was larger than a threshold δ and we studied the impact of δ on link inference. We report the statistical data of true positive instances (TP) and true negative instances (TN), and calculate corresponding rates of TP and TN, as illustrated in Table 1. When threshold $\delta = 2$, the rates of correctly inferred edges are only 9.12% and 25.49% for **Email-1** and **LiveJ-1**, respectively. For both datasets, the rates of correctly inferred edges become higher as δ increases. When $\delta = 18$, TPR in **LiveJ-1** even reaches 86.98%, and 63.61% in **Email-1**. TNR gets slightly decreased when δ increases but still keeps as high as 99%. From our evaluation results in Table 1, we clearly see that the re-identification power of link inference is strong enough to help adversaries identify sensitive relationships with high probability. Hence, link inference attacks are real in practice.

Previous work on protecting privacy of social network does not consider link inference attacks. Link inference associated information can be studied by adversaries, even though the releasing graph is anonymized through existing methods. Given graph G in Fig.1(a), according to existing methods [3,9], we obtain an anonymized graph G_a in Fig.1(c). G_a guarantees the probability of an adversary re-identifying any vertex or edge is at most $\frac{1}{k}$. For G_a, the constant k is 2. Obviously, the expected number of common neighbors between v_0 and v_5 is $1 \times \frac{6656}{14400} + 2 \times \frac{3216}{14400} + 3 \times \frac{416}{14400} + 4 \times \frac{16}{14400} = 1$, and the expected number is $1 \times \frac{252}{864} + 2 \times \frac{18}{864} = \frac{288}{864}$ between v_0 and v_8. However, the adversaries can still infer that the probability of v_0 having a link with v_5 is larger than with v_8, which is the same with previous conclusion that we draw based on original graph G.

Table 1. Re-identification power of link inferences on real social networks

δ	Email-1				LiveJ-1			
	TP	TPR(%)	TN($\times 10^7$)	TNR(%)	TP	TPR(%)	TN($\times 10^7$)	TNR(%)
2	6425	9.12	1.242	99.96	12028	25.49	1.244	99.95
6	3849	26.80	1.248	99.94	6495	57.04	1.247	99.91
10	2568	41.93	1.248	99.93	3896	69.62	1.248	99.89
14	1722	53.76	1.248	99.93	2504	79.14	1.248	99.88
18	1117	63.61	1.249	99.92	1550	86.98	1.248	99.87

1.2 Challenges and Contributions

We address the problem of link inference attacks in social networks. The challenge of our problem lies in the complexity of link inferences in social network. Typically, the adversary can perform inference attack with multiple steps. In order to avoid privacy leakage, we need to modify the social graph. Due to the complexity of graph, a little modification (e.g., remove an edge) may incur a great impact on link inferences. How to cut off the inference paths of sensitive relationships without privacy leakage meanwhile incurring minimal information loss has proposed a great challenge.

Our contributions can be summarized as follows. (1) We discuss two types of link inference attacks, namely, *one-step link inference* attacks and *cascaded link inference* attacks, which have strong link re-identification power in real networks. (2) We propose *inference security* to protect privacy against link inference attacks. (3) We develop a general framework for preventing link inference attacks, which adopts a novel lineage tracing mechanism to efficiently cut off the inference paths of sensitive relationships. (4) We propose algorithms for preventing *one-step link inference* attacks and *cascaded link inference* attacks. (5) We design a number of techniques to make the inference preventing methods efficient meanwhile maintaining the utility. (6) Our extensive empirical studies show that our methods perform well in real datasets.

The remainder of the paper is organized as follows. Related work are summarized in Section 2. In Section 3, we give the problem definition and formalize the inference security model. We outline a general framework for preventing link inference attacks, and propose algorithms for obtaining inference secure graphs in Section 4. We evaluate our methods in Section 5. Section 6 concludes the paper.

2 Related Work

With the increasing popularity of social networks, protecting privacy information in social networks while retaining data utility for data mining and analysis has become an interesting problem that has been studied by a number of recent works. Privacy attacks in social networks are mainly classified into two categories, including *vertex re-identification* attacks and *link re-identification* attacks.

In vertex re-identification attacks (a.k.a. identity disclosure), an adversary identifies the identities of vertices in the published network using the subgraphs associated with target individuals as background knowledge. Liu et al. [1] propose k-degree to prevent from privacy attacking using vertex degree as adversary knowledge. Zhou et al. [2] provide identity privacy through anonymizing the 1-neighborhood subgraph of each vertex. Hay and Campan [3,4] propose to protect identity privacy against subgraph knowledge through clustering vertices into super vertices, where the vertices in a super vertex are indistinguishable from each other. Zou et al. [5] propose a privacy preserving model K-Automorphism for protecting identity privacy.

In link re-identification (i.e., link disclosure) attacks, an adversary aims at identifying sensitive relationships among the individuals in social network. Zheleva et al. [6] discuss a number of privacy preserving strategies to prevent sensitive edge disclosure, in which the protection of link privacy cannot be guaranteed. Ying et al. [7] study graph randomization through adding/removing and switching edges randomly while preserving the spectrum of the network, which do not provide quantifiable privacy protection. Cormode et al. [8] propose a permutation based approach to protect the privacy of links in bipartite graphs. Bhagat et al. [9] improve the work in [8] and study graph anonymization to protect link privacy based on vertices grouping.

Different from considering identity disclosure and link disclosure problems separately, recent work [10,11] has focused on proposing general frameworks to protect both identity and link privacy. Cheng et al. [10] study how to partition and anonymize a releasing graph into disjoint k subgraphs that are isomorphic to each other for ensuring the probabilities of identifying an identity and a sensitive link both at most $\frac{1}{k}$. Besides protecting identity and link privacy, Yuan et al. [11] give a solution to satisfy different needs on privacy protecting level.

One fundamental problem underlying all of the research work mentioned above is that each of them assumes adversaries acquire privacy information using target individual's associated graph structural knowledge, and ignores that it's very possible for adversaries to infer privacy using graph inference knowledge. In this work, we try to protect sensitive relationships in a releasing graph against link inference attacks.

3 Preliminaries and Problem Definition

We model a social network as a simple graph, $G = (V, E)$, where V is the set of vertices, E is the set of edges. We use $V(G)$ and $E(G)$ to refer to the vertex set and edge set of G. In this work, we also use link and relationship interchangeably to denote an edge.

Generally, link inference techniques are classified into three categories, including similarity based link inference, maximum likelihood link inference and probabilistic models based link inference. In this work, we choose common neighbor similarity based link inference (a similarity based link inference technique) as our specific link inference preventing problem.

Definition 1. *(common neighbor similarity) The common neighbor similarity of vertices u and v is defined as the count of common neighbors of u and v, denoted as $Sim_{CN}(u, v)$, which is formalized as Equation 1,*

$$Sim_{CN}(u, v) = |\Gamma(u) \cap \Gamma(v)| \tag{1}$$

where $\Gamma(u)$ refers to the neighbors of u.

We choose common neighbor similarity based link inference (LI_{CN} for short) as adversaries' graph inference knowledge for the following reasons:

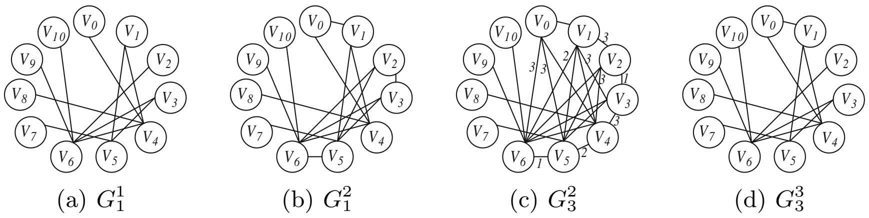
(1) As shown in Table 1, LI_{CN} is simple, effective and has strong link re-identification power, which helps adversaries identify sensitive relationships with high probability;
(2) For adversaries, LI_{CN} is easy to implement. The necessary information for adversaries to perform LI_{CN} are the common neighbor sets of the two target individuals, which are easily collected in real social networks. For instance, in Online Social Networks (OSNs), such as Facebook and MySpace, after the process of sending a friend application to a user and being one of his/her friends, you can access all his/her friends (i.e., neighbors).

Definition 2. *(sensitive edge) Given graph $G(V, E)$, if edge e_{uv} is defined as sensitive by data owners or two involved individuals u and v, then e_{uv} should not exist in G, and vertex pair (u, v) is denoted as sensitive pair.*

In this work, we assume *some* (not all) edges in G are defined as sensitive, and prevent the disclosures of these edges due to link inferences. As shown in Table 1, although an edge is removed, the existence of this edge could be inferred with high probability based on graph inference knowledge.

Definition 3. *(link inference) Given graph $G(V, E)$ and threshold δ, if the common neighbor similarity of vertices u and v satisfies $Sim_{CN}(u, v) \geq \delta$, it is inferred that there exists an edge between u and v.*

For two individuals with a sensitive relationship, the data owner provides a minimum threshold δ to specify his tolerance of revealing closeness of them. Threshold δ can also be personalized by these two involved individuals.



(a) G_1^1 (b) G_1^2 (c) G_3^2 (d) G_3^3

Fig. 2. Cascaded link inferences

Definition 4. *(one-step link inference) Given graph G(V,E) and threshold δ, one-step link inference on G refers to performing link inference on each unconnected vertex pair in G, such that for any unconnected vertex pair (u, v) with $Sim_{CN}(u, v) \geq \delta$, we add e_{uv} to $E(G)$.*

Given graph G in Fig. 1(a) and δ=2, after one-step link inference on G, we obtain G_1 in Fig. 1(d), where $\{e_{v_0v_5}, e_{v_0v_6}, e_{v_1v_2}, e_{v_1v_3}, e_{v_2v_3}, e_{v_2v_4}, e_{v_3v_4}, e_{v_5v_6}\}$ (dotted edges) are newly inferred edges.

Definition 5. *(cascaded link inference) Given graph $G(V, E)$, threshold δ and an integer i, i-times cascaded link inference (i-inference for short) on G refers to performing one-step link inference iteratively on G for i times, and we obtain the i-inference graph of G, denoted as G_i.*

In cascaded link inference, if an edge is inferred from the original graph, this edge is also used for future link inference. Obviously, one-step link inference is the special case of i-inference when $i = 1$, i.e. 1-inference. G_0 denotes graph G with no link inference. $E(G_i) \backslash E(G_{i-1})$(we define G_{-1}=ϕ) refers to the inferred edges in the i-th one-step link inference.

Definition 6. *(inference secure) Given graph $G(V, E)$, sensitive edge set S, an integer i $(i \geq 0)$, and threshold δ, if $S \cap E(G_i) = \phi$, then G is i-inference secure.*

Example 1. Given G in Fig. 1(a), Fig. 1(b) lists three variants of G with some edges removed. After 3-inference on G^2 in Fig. 1(b), we obtain G_3^2 in Fig. 2(c), where edges labeled with integer i $(i = 1, 2, 3)$ are inferred in i-th one-step link inference. For instance, $e_{v_1v_6}$ is inferred in the 2*nd* one-step link inference and belongs to $E(G_2^2) \backslash E(G_1^2)$. Let $S = \{e_{v_0v_5}, e_{v_0v_6}\}$ and $\delta = 2$, since $S \cap E(G_2^2) = \phi$ and $S \cap E(G_3^2) = S$, G^2 is 2-inference secure, not 3-inference secure. Similarly, G is 0-inference secure and G^3 is 3-inference secure.

Theorem 1. *Given graph G, sensitive edge set S, and threshold δ, if $\forall e_{uv} \in S$ satisfies $e_{uv} \notin E(G)$ and $Sim_{CN}(u, v) < \delta$, then G is 1-inference secure.*

Proof. After one-step link inference on G, we obtain G_1. For $\forall e_{uv} \in S$, we have $Sim_{CN}(u, v) < \delta$ and $e_{uv} \notin E(G)$, thus e_{uv} would not exist in $E(G_1)$ due to link inference. Hence, G is 1-inference secure.

Problem 1. (Optimal Inference Security) Given graph G, sensitive edge set S, integer i and threshold δ, find an i-inference secure graph G' with $V(G)=V(G')$ and $E(G)\cap E(G')=E(G')$, such that $|E(G)\setminus E(G')|$ is minimized.

Theorem 2. *The problem of Optimal Inference Security is NP-hard.*

Proof. The proof is by reducing the NP-complete problem of SATISFIABILITY [12]. Limited by space, we omit the details here.

In the next section, we derive a novel lineage tracing mechanism to efficiently cut off inference paths of sensitive relationships meanwhile incurring low information loss.

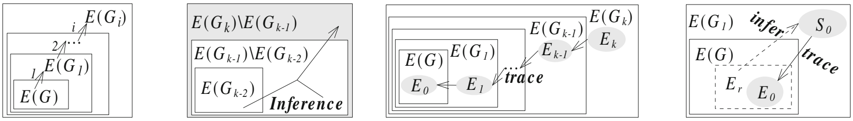
4 Preventing Link Inference Attacks

In this section, we study preventing link inference attacks in social networks. We first outline the general framework for preventing link inference attacks, then propose algorithms for preventing one-step link inference attacks and cascaded link inference attacks.

4.1 A General Framework

Given graph G that is not i-inference ($i\geq 0$) secure, i.e. $S\cap E(G_i)\neq\phi$, we provide lineage tracing mechanism to cut off the inference paths of $S\cap E(G_i)$. Informally, all edges that contribute to the inference of $S\cap E(G_i)$ are considered as the lineage of $S\cap E(G_i)$. The key is to efficiently find inference paths of $S\cap E(G_i)$.

Fig.3(a) describes the process of obtaining G_i, where the arrow line labeled with integer k (k=1,. . .,i) refers to the k-th one-step link inference. Clearly, edges in $E(G_k)\setminus E(G_{k-1})$ are due to the combinational impact of $E(G_{k-1})\setminus E(G_{k-2})$ and $E(G_{k-2})$, as depicted in Fig.3(b). Intuitively, we can prevent inferring edges in $E(G_k)\setminus E(G_{k-1})$ through removing edges in $E(G_{k-1})\setminus E(G_{k-2})$. We formalize this intuition in Theorem 3.



(a) Obtain G_i (b) Union infer (c) Lineage tracing (d) Prevent S_0

Fig. 3. Cutting off inference paths through lineage tracing

Theorem 3. *Given graph $G(V,E)$, edges in $E(G_k)\setminus E(G_{k-1})$ can be prevented from inferring through removing edges in $E(G_{k-1})\setminus E(G_{k-2})$.*

Proof. (Proof by Contradiction.) Assume to the contrary that there exists an edge e in $E(G_k)\backslash E(G_{k-1})$ that cannot be prevented from inferring through removing edges in $E(G_{k-1})\backslash E(G_{k-2})$, i.e. edge e can still be inferred after removing $E(G_{k-1})\backslash E(G_{k-2})$ from G_{k-1}. Thus, e can be inferred based on $E(G_{k-2})$ through one-step link inference and e is obviously in $E(G_{k-1})\backslash E(G_{k-2})$, contradicting our assumption that e is in $E(G_k)\backslash E(G_{k-1})$.

As shown in Fig. 3(c), given graph G and edge set $E_k \subseteq E(G_k)\backslash E(G_{k-1})$, Theorem 3 inspires us that we can prevent inferring E_k by removing edges in $E(G_{k-1})\backslash E(G_{k-2})$, noted as E_{k-1}. Similarly, we prevent inferring E_{k-1} by removing $E_{k-2} \subseteq E(G_{k-2})\backslash E(G_{k-3})$. We perform these lineage tracing and removing operations iteratively until we remove E_0 in $E(G)$. After above operations, edges in E_k would not exist in $E(G_k)$.

Clearly, for graph G that is $(k-1)$-inference secure but not k-inference secure (i.e., $S \cap E(G_{k-1}) = \phi$ and $S \cap E(G_k) \neq \phi$), we can prevent inferring $S \cap E(G_k)$ using lineage tracing and removing operations described in Fig. 3(c) and obtain k-inference security for G. Given graph G and an integer i ($i \geq 0$), if we want to obtain i-inference security for G, we can firstly obtain 0-inference secure graph (i.e. remove sensitive edges directly from $E(G)$), and then obtain k-inference (k=1,. . .,i) secure graph based on $(k-1)$-inference secure graph iteratively.

Based on the framework, we propose algorithms for preventing one-step link inference attacks and cascaded link inference attacks in the following subsections, where we will elaborate the technical details of lineage tracing and removing.

4.2 Preventing One-Step Link Inference Attacks

Algorithm 1 protects input graph G against one-step link inference attacks (i.e., to obtain 1-inference secure graph), the process of which is outlined in Fig. 3(d). The fact that the graph produced by Algorithm 1 is 1-inference secure is a straightforward result of Theorem 1. The key idea is to make Sim_{CN} of each sensitive edge less than δ.

Algorithm 1 firstly removes sensitive edges from $E(G)$ to obtain 0-inference security (Line 1). Then S_0 is initialized with sensitive edges with $Sim_{CN} \geq \delta$ (Line 2). All edges present in S_0 would be in $E(G_1)$. Removable edge set E_r that contribute to the inference of S_0 is generated (Lines 3-5). In practice, we can obtain 1-inference secure graph through removing $E_0 \subseteq E_r$ from $E(G)$ and E_0 should contain as less edges as possible. In procedure **Remove_Edge**, we provide different strategies for removing edges in E_r (Line 7). While removing edges in E_r, sensitive edges already with $Sim_{CN} < \delta$ should be excluded from S_0 (Line 8). In order to minimize the changes on graph properties due to edge removing, for a removed edge e, the procedure **Find_Add_Edge** finds a new edge e' to add to $E(G)$ (Line 9). Hence, the condition of $E(G) \cap E(G') = E(G')$ in Problem 1 is relaxed to $E(G) \cap E(G') \approx E(G')$. Algorithm 1 performs the procedures **Remove_Edge** and **Find_Add_Edge** repeatedly until graph G is 1-inference secure. We present the details of Algorithm 1 in the followings.

Algorithm 1: Preventing One-Step Link Inference Attacks

```
   Input: Graph $G(V, E)$, sensitive edge set $S$ and threshold $\delta$
   Output: 1-inference secure graph $G$
 1 $E(G) \leftarrow E(G) \backslash S$ ;                      /* Remove sensitive edges in $G$ */
 2 $S_0 \leftarrow \{e_{uv} | \forall e_{uv} \in S \& Sim_{CN}(u, v) \geq \delta\}$ ;   /* Initialize $S_0$ */
 3 for each $e_{uv} \in S_0$ do                               /* Generate $E_r$ */
 4     for each $w \in \Gamma(u) \cap \Gamma(v)$ do
 5         Add $e_{uw}, e_{wv}$ to $E_r$;
 6 repeat
 7     $e \leftarrow$ Remove_Edge($S_0, E_r$) ;               /* Remove edge $e$ in $E_r$ */
 8     update $S_0$;
 9     $e' \leftarrow$ Find_Add_Edge($e$) ;        /* Based on $e$, find a new edge $e'$ */
10     if new edge $e'$ for $e$ is found then
11         $E(G) \leftarrow E(G) \cup \{e'\}$;
12 until $S_0$ is empty ;
13 return $G$;
```

4.2.1 Generate Removable Edge Set

When we generate removable edge set in Algorithm 1 (Lines 3-5), we only consider edges that contribute to the Sim_{CN} of sensitive edges in S_0. As shown in Algorithm 1, S_0 only contain sensitive edges with $Sim_{CN} \geq \delta$.

Definition 7. *(removable edge set) Given graph G, sensitive edge set S, and threshold δ, we use removable edge set E_r to denote the set of all edges that connect between vertices of sensitive pairs with $Sim_{CN} \geq \delta$ and their common neighbors.*

Example 2. Given graph G in Fig.1(a), let $S = \{e_{v_0v_5}, e_{v_0v_6}\}$ and $\delta = 2$. Since $Sim_{CN}(v_0, v_5)$ and $Sim_{CN}(v_0, v_6)$ both equal to $3 > \delta$, we obtain removable edge set $E_r = \{e_{v_0v_1}, e_{v_0v_2}, e_{v_0v_3}, e_{v_0v_4}, e_{v_1v_5}, e_{v_2v_5}, e_{v_3v_5}, e_{v_2v_6}, e_{v_3v_6}, e_{v_4v_6}\}$.

Note that there always exists a feasible solution to obtain a 1-inference secure graph. In the worst case, all edges in E_r can be removed, i.e. remove $E_0{=}E_r$ from $E(G)$. In this way, Sim_{CN} of all sensitive pairs equal to 0; thus, any inference security requirement is satisfied. However, we want to remove *minimum* edges in E_r to obtain a 1-inference secure graph. In the next subsection, we introduce our heuristic strategy of removing edges in E_r.

4.2.2 Remove Edges in E_r

Taking S_0 and E_r as input, **Remove_Edge** removes an edge in E_r and return it. We provide different strategies of removing edges in E_r.

A naive strategy of **Remove_Edge** is that for sensitive edge $e_{uv} \in S_0$ with $Sim_{CN}(u, v) \geq \delta$, we randomly remove $(Sim_{CN}(u, v) - \delta + 1)$ edges in E_r. The removed edges must meet the following conditions: (1) These edges connect between u (or v) and vertices in $\Gamma(u) \cap \Gamma(v)$; (2) None of these edges connect to the same common neighbor. Removing such edges incurs $Sim_{CN}(u, v) < \delta$.

After performing such removing operation on each sensitive edge in S_0, G is 1-inference secure. Since $S_0 \subseteq S$ and $\delta \geq 1$, the computational complexity of naive strategy is $\mathcal{O}(|S|Sim_{max})$, where Sim_{max} denotes the maximal Sim_{CN} of the sensitive edges in S.

In order to prevent inferring S_0 with minimum edges removed in E_r, we design a heuristic function IC(Inference Contribution) to evaluate the contribution of each edge in E_r for the inference of S_0, which is formalized as Equation 2.

$$IC(e_{uv}) = countif_{m \in \Gamma(v)}(e_{um} \in S_0) + countif_{n \in \Gamma(u)}(e_{nv} \in S_0) \tag{2}$$

Clearly, removing the edge with larger IC value results in more decrease on Sim_{CN} of sensitive edges in S_0. A heuristic strategy of **Remove_Edge** is to always remove the edge in E_r with the largest IC. Obviously, heuristic strategy remove at most $|S|Sim_{max}$ edges in E_r. Since removing an edge introduces $|E_r|$ operations to search the edge with the largest IC in E_r and there are at most $2|S|Sim_{max}$ edges in E_r, the computational complexity of the heuristic strategy is $\mathcal{O}(2|S|^2Sim_{max}^2)$.

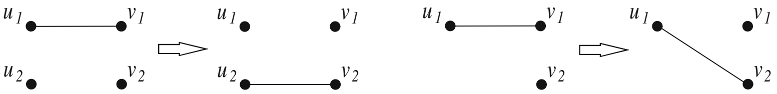
Considering Example 2, for graph G, we obtain G^1 and G^2 in Fig.1(b) with naive and heuristic strategy, respectively. As shown in Fig.2, G^1 and G^2 are both 1-inference secure. However, G^1 incurs more edges removed than G^2.

4.2.3 Add Edges for Preserving Graph Properties

Although existing research work [7] studies graph randomizing techniques meanwhile preserving graph spectrum, it is not realizable to adopt the methods in [7] when the amount of edge modifications is large. We design two efficient strategies on adding new edges for preserving graph properties.

Definition 8. *(Non Inference Contributing Edge) Given graph G, sensitive edge set S, and threshold δ, for an edge $e \notin E(G)$, if adding e to $E(G)$ does not change S_0=$\{e_{uv}|\forall e_{uv} \in S \& Sim_{CN}(u,v) \geq \delta\}$, then edge e is a non inference contributing edge, denoted as nic-edge for simplicity.*

Notice that the procedure **Find_Add_Edge** only consider nic-edges as candidates to ensure new edges would not incur changes of S_0.



(a) Deletion/Addition (b) Switch

Fig. 4. Add new edges for preserving graph properties

Deletion/Addition. The first edge adding strategy is `Deletion/Addition`, which is illustrated in Fig.4(a). When edge $e_{u_1v_1}$ is removed from $E(G)$, we add new edge $e_{u_2v_2}(e_{u_2v_2} \notin E(G))$ to $E(G)$, which is with the largest Structural

Similarity (SS) to $e_{u_1v_1}$. Structural Similarity (SS) is formalized as Equation 3. For a removed edge $e_{u_1v_1}$, there exist at most $\frac{|V(G)|(|V(G)|-1)}{2}-|E(G)|$ new edge candidates.

$$SS(e_{u_1v_1}, e_{u_2v_2}) = \frac{|(\varGamma(u_1) \cup \varGamma(v_1)) \cap (\varGamma(u_2) \cup \varGamma(v_2))|}{|(\varGamma(u_1) \cup \varGamma(v_1)) \cup (\varGamma(u_2) \cup \varGamma(v_2))|} \tag{3}$$

Switch. Fig.4(b) describes another efficient new edge adding strategy, named as `Switch`. Different from `Deletion/Addition`, in `Switch`, when edge $e_{u_1v_1}$ is removed, we add new edge that connects between one vertex in $\{u_1, v_1\}$ and other vertices in $V(G)$. For instance, as shown in Fig.4(b), edge $e_{u_1v_1}$ is removed and new edge $e_{u_1v_2}$ is added to $E(G)$. When adopting `Switch`, Structural Similarity $SS(e_{u_1v_1}, e_{u_1v_2})$ is simplified into $SS(e_{u_1v_1}, e_{u_1v_2})=\frac{|\varGamma(u_1)\cup(\varGamma(v_1)\cap\varGamma(v_2))|}{|\varGamma(u_1)\cup(\varGamma(v_1)\cup\varGamma(v_2))|}$. Obviously, for a removed edge $e_{u_1v_1}$, there exist at most $(2(|V(G)|-2))$ new edge candidates in `Switch`, which is less than `Deletion/Addition`.

Although `Switch` achieves a better efficiency than `Deletion/Addition` through considering less new edge candidates, it performs well in preserving graph properties, which will be shown in the experimental section.

4.3 Avoiding Cascaded Link Inference Attacks

Based on the general framework, we now propose the algorithm of avoiding cascaded link inference attacks (i.e., transform a graph to be i-inference secure), which is shown in Algorithm 2.

Algorithm 2: Avoiding Cascaded Link Inference Attacks

```
  Input: Graph $G(V, E)$, threshold $\delta$, sensitive edge set $S$ and an integer $i$
  Output: $i$-inference secure graph $G$
1 $E(G) \leftarrow E(G) \backslash S$ ;                     /* Make $G$ 0-inference secure */
2 for $k = 1$ to $i$ do        /* Obtain $k$-inference secure graph iteratively */
3     $S_0 \leftarrow E(G_k) \cap S$ ;                /* Obtain sensitive edges in $G_k$ */
4     for $j=k-1$ to $0$ do                  /* Lineage tracing and removing */
5         $E_r \leftarrow \phi, E_s \leftarrow \phi$;
6         generate $E_r$ for $S_0$ using edges in $E(G_j) \backslash E(G_{j-1})$;
7         remove $E_s \subseteq E_r$ to prevent inferring $S_0$;
8         $S_0 \leftarrow E_s$;
9 return $G$;
```

Algorithm 2 firstly make G 0-inference secure (Line 1), then obtains k-inference ($1 \le k \le i$) secure graph iteratively (Lines 2-8). When transform a graph from ($k-1$)-inference secure into k-inference secure (Lines 3-8), we firstly obtain sensitive edges inferred in G_k (Line 3), i.e. $E(G_k) \cap S$. Then, we perform lineage tracing and removing operations iteratively to cut off the inference paths of $E(G_k) \cap S$ (Lines 4-8). For each iteration with j, S_0 contains the removed edges in $E(G_{j+1})\backslash E(G_j)$, and we remove edges in $E(G_j)\backslash E(G_{j-1})$ to prevent inferring S_0. We adopt the general idea of Algorithm 1 to prevent inferring S_0, and make some modifications and extensions to original method. Firstly, when we generate E_r that contribute to the inference of S_0, we only consider edges in $E(G_j)\backslash E(G_{j-1})$ (Lines 6). Secondly, Sim_{CN} of edges in S_0 are calculated on $E(G_j)$, and edges in $E(G_k)\backslash E(G_j)$ are neglected (Line 7).

Now, we analyze the computational complexity of Algorithm 2. When tracing lineage in $E(G_j)\backslash E(G_{j-1})$, E_r and E_s contain at most $2|S|Sim_{max}^{k-j}$ and $|S|Sim_{max}^{k-j}$ edges respectively, where Sim_{max} denotes the maximal Sim_{CN} of the edges in $E(G_k)$. Algorithm 2 with naive edge removing strategy removes $\Sigma_{k=1}^{i}\Sigma_{j=k-1}^{0}|S|Sim_{max}^{k-j} \leq |S|Sim_{max}^{i}$ edges for obtaining i-inference security. Hence, the computational complexity of Algorithm 2 with naive edge removing strategy is $\mathcal{O}(|S|Sim_{max}^{i})$, where Sim_{max} is the maximal Sim_{CN} of the edges in $E(G_i)$. Similarly, the computational complexity is $\mathcal{O}(2|S|^2Sim_{max}^{2i})$ for Algorithm 2 with heuristic edge removing strategy.

5 Experimental Evaluation

In this section, we provide extensive experiments to evaluate our methods. We use two real network datasets, **Email-1** and **LiveJ-1**, which are also used in [10]. There are 5000 vertices and 11047 edges in **Email-1** with average degree 4.42, 5000 vertices and 17847 edges in **LiveJ-1** with average degree 7.14, respectively. Table 2 lists the statistics of vertex pairs with Sim_{CN}=1,. . .,10 in each dataset.

Table 2. Statistics of Sim_{CN} in networks

Dataset	Common Neighbor Similarity (Sim_{CN})									
	1	2	3	4	5	6	7	8	9	10
Email-1	164144	31146	13068	7307	4558	3075	2153	1723	1285	1014
LiveJ-1	284567	21158	7510	4363	2771	2028	1575	1220	967	792

We implement four versions of Algorithm 1 for preventing one-step link inference attacks, which are *Naive*(`N`, `NA`), *LIP*(`H`, `NA`), *D/A-LIP*(`H`, `D/A`) and *S-LIP*(`H`, `S`), where `N` and `H` refer to remove edges with naive and heuristic strategy respectively, `NA` refers to no edge addition, and `D/A` and `S` refer to edge addition with `Deletion/Addition` and `Switch` strategy respectively. We also implement Algorithm 2 for avoiding cascaded link inference attacks, named as $CLIP$, which adopts heuristic edge removing strategy. All the programs are implemented in Java. The experiments are performed on a 2.33GHz Intel Core 2 Duo CPU with 4GB DRAM running the Windows XP operating system.

5.1 Performance of Link Inference Preventing v.s. Sim_{CN}, δ

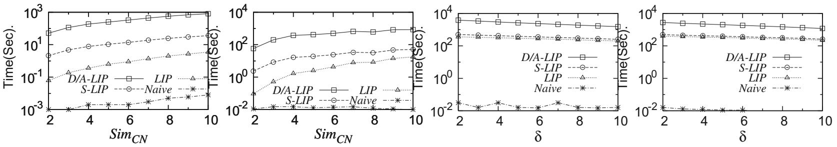
We design two set of experiments to evaluate the impacts of Sim_{CN} and δ on the performance of one-step link inference preventing algorithms. Firstly, for evaluating the impact of Sim_{CN}, we set δ=2 and generate S through randomly sampling 200 unconnected vertex pairs with Sim_{CN}=2,…,10 in **Email-1** and **LiveJ-1**, respectively. Secondly, for evaluating the impact of δ, we generate S with unconnected vertex pairs with $Sim_{CN} \geq 15$ and obtain 1193 and 521 sensitive pairs in **Email-1** and **LiveJ-1** respectively, and set δ=2,…,10.

5.1.1 Runtime and Information Loss

We use $\mathcal{I}.\mathcal{L}. = \frac{number\ of\ removed\ edges}{|E(G)|}$ to evaluate information loss of inference preventing algorithms. Fig.5 and Fig.6 show the results of runtime and information loss. Due to the same edge removing strategy for *LIP*, *D/A-LIP* and *S-LIP*, we only plot *LIP* in Fig.6 as representative. Generally, as δ is constant and Sim_{CN} increases, both runtime and information loss get higher as shown in Figs.5(a), 5(b) and Figs.6(a), 6(b), respectively. The reason is intuitively that preventing the inferences of sensitive edges with larger Sim_{CN} incurs more edges to remove. An exception arises in Fig.6(b) when Sim_{CN}=8, where the algorithms incurs less information loss than Sim_{CN}=7, which seems unexpected. However, on closer inspection, we find that the average IC of edges in E_r is 2.911 when Sim_{CN}=8, which is much higher than 2.294(Sim_{CN}=7) and 2.443(Sim_{CN}=9). Such observation shows that the IC value is an important factor affecting the performance of the link inference preventing algorithms. When S is constant and δ gets increased, the algorithms require lower runtime and incur less information loss, which are depicted in Figs.5(c),5(d) and Figs.6(c), 6(d), respectively, since higher δ refers to less inference secure requirement. In Fig.5, the runtime of the algorithms can be summarized as *Naive*≪*LIP*<*S-LIP*≪*D/A-LIP*. With our heuristic edge removing strategy, the information loss of *LIP* is much lower than *Naive*, as shown in Fig.6.

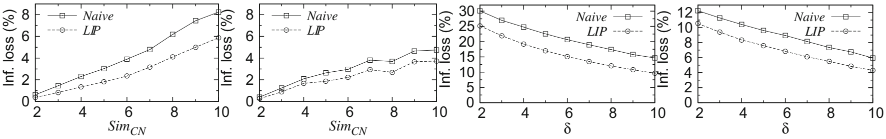
5.1.2 Data Utilities

We examine two graph structural properties, *Transitivity* and *Average Path Length* (see [13] for details), on 1000 vertices randomly sampled in vertices of S and their neighbors. We use `Change ratio` $= |P_o - P_s|/|P_o|$ to evaluate the property change ratio, where P_o and P_s refer to the property values of the original graph G and the inference secure version of G. Fig.7 and Fig.8 show the change ratios of *Transitivity* and *Average Path Length*, respectively. Generally, *S-LIP* and *D/A-LIP* are most effective in terms of preserving graph properties, and their change ratios are around 2% for **Email-1** and 7% for **LiveJ-1**. The curves of *Naive* and *LIP* in Fig.7 and Fig.8 behave similarly to the ones in Fig.6. Although considering less adding edge candidates, in practice, *S-LIP* performs as well as *D/A-LIP* but with higher efficiency. We notice that *S-LIP* and *D/A-LIP* do not preserve graph properties of **LiveJ-1** as well as **Email-1**. Such observation could be explained by the statistic data in Table 2, where except for $Sim_{CN} = 1$, the numbers of vertex pairs in **Email-1** with Sim_{CN}=2,…,10 are much higher than in **LiveJ-1**. Hence, the newly added edges in **Email-1** are with higher SS values and preserve graph properties better.



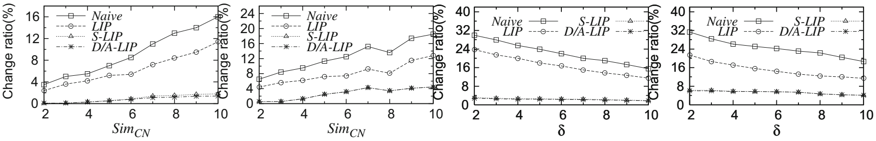
(a) Email-1 (b) LiveJ-1 (c) Email-1 (d) LiveJ-1

Fig. 5. Runtime of preventing one-step link inference



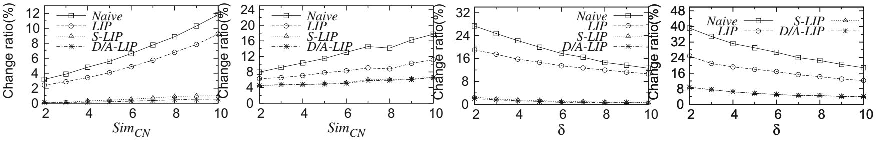
(a) Email-1 (b) LiveJ-1 (c) Email-1 (d) LiveJ-1

Fig. 6. Information loss of preventing one-step link inference



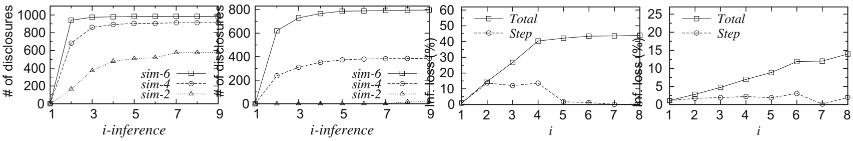
(a) Email-1 (b) LiveJ-1 (c) Email-1 (d) LiveJ-1

Fig. 7. Change ratio of Transitivity



(a) Email-1 (b) LiveJ-1 (c) Email-1 (d) LiveJ-1

Fig. 8. Change ratio of Average Path Length



(a) Email-1 (b) LiveJ-1 (c) Email-1 (d) LiveJ-1

Fig. 9. Re-identification power and information loss in *CLIP*

5.2 Re-identification Power and Information Loss in *CLIP*

Firstly, we evaluate the re-identification power of cascaded link inference. We set δ=8, and generate S through randomly sampling 1000 unconnected vertex pairs with Sim_{CN}=2,4,6 in each dataset. We count the number of disclosures in the i-inference (i=1,. . .,9) graph and show the results in Fig. 9(a) and Fig. 9(b). Overall, the number of disclosures gets higher as i increases and tends to stabilize after i=4. Such observation can be explained by the community theories of real social networks. The social networks consist of a large amount of communities, and the probability of there existing a link between two vertices within a community is much higher than two vertices belong to different communities. Hence, after performing one-step link inference several times, cascaded link inference has disclosed the sensitive edges connecting vertices within a community rather than other ones bridging different communities, such that i-inference (i>4) do not lead to an observable increase in disclosures in any dataset. Specially, for sensitive pairs with Sim_{CN}=6, 4-inference causes 100% and 80% of these edges disclosed in **Email-1** and **LiveJ-1**, respectively. Hence, cascaded link inference is indeed a privacy threat for sensitive relationships in real networks.

Secondly, we examine the information loss of *CLIP*. We set δ=3, and generate S through randomly sampling 200 unconnected vertex pairs with Sim_{CN}=4 in each dataset. *Total* and *Step* in Figs. 9(c), 9(d) refer to the information loss for obtaining i-inference secure graph and transforming a (i−1)-inference secure graph into i-inference secure, respectively. As depicted in Fig. 9(c) and Fig. 9(d), *Total* gets higher as i increases from 1 to 8 meanwhile *Step* behaves unsteadily. An interesting observation of *Step*, namely *Sharp Drop*, arises in **Email-1** when i=5 and in **LiveJ-1** when i=7, where *Step* decreases sharply. Such observation could be interpreted as follows. Before *Sharp Drop* arises, edges in G are gradually removed by *CLIP*. When most of the inference paths of S have been cut off, *CLIP* would remove much less edges in G than before to obtain an i-inference secure graph, which is a *Sharp Drop*. However, inference paths could be reconstructed by cascaded link inference and *Step* may increase after *Sharp Drop*, as shown in Fig. 9(d) when i=8. Overall, studying the inference paths of sensitive links is the key to avoid cascaded link inference attacks.

6 Conclusion

In this paper, we discuss an important privacy problem in social networks, namely *link inference attacks*. We formalize *inference security* and develop a general framework for obtaining inference secure graphs. We propose efficient algorithms for preventing *one-step link inference* attacks and *cascaded link inference* attacks. An extensive empirical study on real datasets indicates that link inference attacks are real in practice, and our methods perform well in terms of privacy protection and efficiency meanwhile maintaining graph properties.

References

1. Liu, K., Terzi, E.: Towards identity anonymization on graphs. In: SIGMOD, pp. 93–106 (2008)
2. Zhou, B., Pei, J.: Preserving privacy in social networks against neighborhood attacks. In: ICDE, pp. 506–515 (2008)
3. Hay, M., Miklau, G., Jensen, D., Towsley, D.: Resisting structural re-identification in anonymized social networks. In: VLDB, pp. 102–114 (2008)
4. Campan, A., Truta, T.M.: A clustering approach for data and structural anonymity in social networks. In: PinKDD (2008)
5. Zou, L., Chen, L., Ozsu, M.T.: K-automorphism: A general framework for privacy preserving network publication. VLDB Endowment 2(1), 946–957 (2009)
6. Zheleva, E., Getoor, L.: Preserving the Privacy of Sensitive Relationships in Graph Data. In: Bonchi, F., Malin, B., Saygın, Y. (eds.) PInKDD 2007. LNCS, vol. 4890, pp. 153–171. Springer, Heidelberg (2008)
7. Ying, X., Wu, X.: Randomizing social networks: a spectrum preserving approach. In: SDM, pp. 739–750 (2008)
8. Cormode, G., Srivastava, D., Yu, T., Zhang, Q.: Anonymizing bipartite graph data using safe groupings. VLDB Endowment 1(1), 833–844 (2008)
9. Bhagat, S., Cormode, G., Krishnamurthy, B., Srivastava, D.: Class-based graph anonymization for social network data. VLDB Endowment 2(1), 766–777 (2009)
10. Cheng, J., Fu, A.W.-C., Liu, J.: K-Isomorphism: Privacy preserving network publication against structural attacks. In: SIGMOD, pp. 459–470 (2010)
11. Yuan, M., Chen, L., Yu, P.S.: Personalized privacy protection in social networks. VLDB Endowment 4(2), 141–150 (2010)
12. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC, pp. 151–158 (1971)
13. Costa, L.D.F., Rodrigues, F.A., Travieso, G., Boas, P.R.V.: Characterization of complex networks: A Survey of measurements. Advances in Physics 56(1), 167–242 (2007)

*Y*ou *C*an *W*alk *A*lone: Trajectory Privacy-Preserving through Significant Stays Protection

Zheng Huo[1], Xiaofeng Meng[1], Haibo Hu[2], and Yi Huang[1]

[1] School of Information, Renmin University of China, Beijing, China
{huozheng,xfmeng,westyi}@ruc.edu.cn
[2] Department of Computer Science, Hong Kong Baptist University, Hong Kong
haibo@comp.hkbu.edu.hk

Abstract. Publication of moving objects' everyday life trajectories may cause serious personal privacy leakage. Existing trajectory privacy-preserving methods try to anonymize *k* whole trajectories together, which may result in complicated algorithms and extra information loss. We observe that, background information are more relevant to where the moving objects really visit rather than where they just pass by. In this paper, we propose an approach called *Y*ou *C*an *W*alk *A*lone (*YCWA*) to protect trajectory privacy through generalization of stay points on trajectories. By protecting stay points, sensitive information is protected, while the probability of whole trajectories' exposure is reduced. Moreover, the information loss caused by the privacy-preserving process is reduced. To the best of our knowledge, this is the first research that protects trajectory privacy through protecting significant stays or similar concepts. At last, we conduct a set of comparative experimental study on real-world dataset, the results show advantages of our approach.

Keywords: Privacy-preserving, Trajectory data publication, Stay points extraction.

1 Introduction

Recent years, positioning techniques and location-aware devices have made numerous locations and traces of moving objects (MOBs) collected and published. Mining and analyzing trajectories is beneficial to multiple novel applications. For example, analyzing trajectories of passengers in an area can help people to make commercial decisions, such as where to build a restaurant; while, analyzing trajectories of vehicles in a city may help government to optimize traffic management systems. Although publishing trajectories is beneficial to mobility-related decision making processes, it still causes serious threats to personal privacy: spatio-temporal information contained in trajectories may reveal individuals personal information, such as, living habits, health conditions, social customs, work and home addresses, etc. When we say *trajectory privacy-preserving*, we mean to protect both whole trajectories not to be re-identified and the frequent/sensitive location samples not to be exposed. To address these problems, trajectory *k*-anonymity is proposed to anonymize *k* trajectories together in a broader similar time span [1,2,3]. But we argue that, it is not necessary to involve all location samples into privacy strategy.

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 351–366, 2012.
© Springer-Verlag Berlin Heidelberg 2012

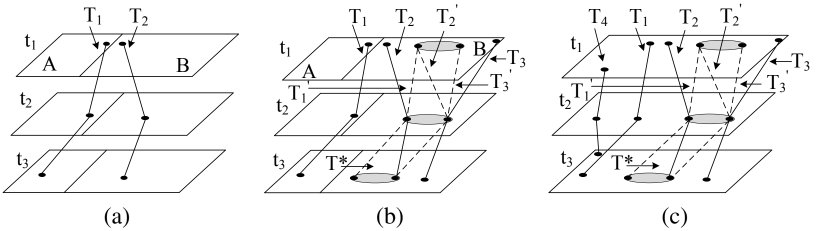


Fig. 1. An example of trajectory *k*-anonymity

Most of the trajectory *k*-anonymity methods try to anonymize *k* whole trajectories together, which may lead to serious information loss. Examples of trajectory *k*-anonymity are shown in Fig.1, where *k*=3. Without loss of generality, number of trajectories n_t in each sub figure is set to 2, 3, 4. They stand for n_t is less than *k*, exactly equals to *k* and larger than *k* respectively. We take (k,δ)-anonymity [2] as an example. When n_t=2 in Fig.1(a), trajectory 3-anonymity cannot be achieved, both trajectories should be deleted for privacy-preserving purpose. Note that, the deletion happens for all $n_t < k$. In Fig.1(b), $n_t = 3$. For a given radius δ, trajectory 3-anonymity can be achieved by trajectory clustering and space translation. Thus, original trajectories T_1, T_2 and T_3 (represented by the solid lines) are translated to T_1', T_2' and T_3' (represented by the dotted lines) respectively, then each location sample is generalized in T^* (represented as the cylinder in gray). Besides generalization, space translation also causes information loss. For example, given a query *Find me trajectories in area A*, it returns nothing if executes on T^*. While in fact, T_1 is in *A*, thus the query result is totally lost. If trajectory (k, δ)-anonymity tries to avoid space translation, anonymity region should be expanded, which may also cause information loss. In Fig.1(c), T_4 is added. However, the radius of the 4 trajectories is too large to be anonymized together, T_4 should be deleted. Then T_1, T_2 and T_3 are anonymized in T^*, the same as in Fig.1(b). Compared with original trajectories, deletion, space translation and generalization are adopted to achieve trajectory *k*-anonymity, while each of them may lead to information loss. Thus, the total information loss of trajectory *k*-anonymity is high.

Instead of treating location samples equally in trajectory *k*-anonymity, our key observation is that real trajectories are not randomly sampled spatio-temporal points, they have semantics, such as *stay points*. We therefore propose to protect trajectory privacy through protecting stay points, which may avoid serious information loss as well as providing high privacy guarantee. This idea is feasible for two main reasons: firstly, most background information is relevant to stay points (*e.g., check-ins at semantic places or credit card transactions at shopping malls*). Thus, protecting stay points may reduce the probability of whole trajectory exposure; secondly, protection of stay points can prevent leakage of sensitive information on trajectories, since stay points contain more sensitive information than ordinary location samples (*e.g., if a MOB visits or stays at a hospital, adversaries may infer the person has a health problem; while the adversary may infer nothing if the MOB just passes through in front of a hospital*). Moreover, trajectory *k*-anonymity highly depends on distributions of MOBs. If the distribution is too sparse to satisfy *k*-anonymity, trajectories may be deleted for privacy-preserving purpose (as shown in Fig.[1]); while if the distribution is too dense, locations or trajectories may be exposed. This is because people always gather in semantic places, if we anonymize k MOBs at a semantic place, their locations are exposed. Our proposal can avoid this by generalizing stay points into *zones*, each zone contains at least l semantic places. Thus, adversaries cannot distinguish MOBs′ exact locations.

In this paper, we study the problem of protecting trajectory privacy in a data publication perspective. The key challenges of our proposal are how to extract stay points efficiently on people's trajectories and how to generate zones with minimized size and diversified contents. The contributions of this paper are as follows:

- We propose to depersonalize significant stay points on trajectories instead of current whole trajectory anonymization.
- We then implement this notion in our proposed method *Y*ou *C*an *W*alk *A*lone (*YCWA*) through splitting trajectories into {move, stay} sequences, and generalizing each stay point into a territory based on a generated split map.
- Two approaches are proposed to generate the split map. One of which is grid-based approach; the other one is clustering-based approach. The latter takes both spatial distance and semantic similarities into consideration.
- We experimentally evaluate the proposed approach on a real-world dataset. Experiment results show that information loss caused by *YCWA* exhibits lower than 20%, which obviously dominates trajectory k-anonymity method.

The rest of the paper is organized as follows. Section 2 summarizes related work. Section 3 formally defines concepts that we study in this paper. In section 4, we present our proposed approach. Section 5 analyzes the privacy guarantee and data utility. Experiment results are shown in section 6. Finally, section 7 concludes the paper.

2 Related Work

Trajectory privacy-preserving is a new research area that has received lots of concerns recent years. Several approaches have been proposed to tackle the problem in a data publication perspective in an off-line manner, while some have been proposed in the context of location-based services in an online manner.

We first introduce privacy-preserving techniques in trajectory data publication. In [2], Abul et al. propose a concept called (k, δ)-anonymity due to the imprecision of GPS devices, where δ represents the possible location imprecision. Then an approach called Never Walk Alone (*NWA*) is proposed to achieve (k, δ)-anonymity through trajectory clustering and space translation. In [3], Yarovoy et al. observe the fact that there does not exist a fixed set of QID attributes for all the moving objects, and the anonymity groups may not be disjoint. A notion of attack graph-based k-anonymity is proposed. Yarovoy et al. propose two algorithms called Extreme Union and Symmetric Anonymization to generate anonymity groups which satisfy the novel k-anonymity. In [1], Nergiz et al. argue that since the trajectories are published for research purpose, it is useful to publish atomic trajectories rather than anonymized regions. So the authors design a method to publish atomic trajectories in an anonymization-reconstruction manner: first, enforce *k*-anonymity by clustering trajectories together based on *log cost distance*, then reconstruct trajectories by randomly selecting location samples from anonymized regions. Privacy strategies in [4] is based on the assumption that different adversaries may have different parts of MOBs' trajectories, while the data publisher knows what the attackers own. Then a suppression-based method is proposed to suppress trajectory segments which may reduce the probability of disclosing whole trajectories. In [6], the authors propose a new trajectory privacy-preserving method which is implemented through spatial generalization and *k*-anonymity.

Recently, several approaches have been proposed to protect MOBs' trajectory privacy in an online manner. In [10], a suppression-based method is proposed to protect users' online trajectory privacy. In this paper, areas are classified as either *sensitive* or *insensitive* based on the proportion of visitors and the whole population of that area. Location updates are suppressed when users enter a sensitive area. In [13], Toby et al. propose to anonymize historical trajectories with users' current trajectories. This proposal helps to reduce the area size of the cloaking region. In [5] Gyozo et al. propose a notion of trajectory privacy-preserving data collection, then implement it based on a server-client architecture. Each client's trajectory is split, exchanged and anonymized by the server before collected by the service provider.

The main difference between these works and our proposal is that they do not account for any difference of location samples, while in fact, stay points are more important and more sensitive than ordinary location samples. As far as we have investigated, we are the first to propose protecting trajectory privacy through protecting significant stay points. Another important concern in this paper is the diversity of sensitive attributes. If the places contained in a zone are almost the same type, the visitors' personal privacy may be exposed. *e.g., if someone visits a zone that only contains a certain type of sensitive locations, such as clubs, even the stay point is generalized to an area, adversaries may still discover he has visited a club no matter which one it is*. We solve this problem by using a mixed distance in the clustering algorithm to enforce diversified places into a zone.

3 Problem Statements

Trajectories of moving objects are collected and stored in moving object databases (MOD). For a moving object O_i, its trajectory $\mathbb{T}$ is a set of discrete locations at sampling time, represented as: $\mathbb{T} = \{q_i, (x_1, y_1, t_1), (x_2, y_2, t_2), \ldots (x_n, y_n, t_n)\}$, where q_i is the identifier of the trajectory; (x_i, y_i) represents MOB's position at sampling time t_i, (x_i, y_i, t_i) is a location sample on trajectories. Raw trajectories consist of GPS record, which is defined in [8]. In the next, we give some definitions in our study.

Definition 1 (Location). *A location $\mathbb{L}$ is a two-tuple $<\boldsymbol{x}, \boldsymbol{y}>$, which represents the latitude and longitude of the location.*

Each GPS record corresponds to a location at sampling time. Location samples are basically in two categories: pass-by points and stay points. Pass-by points are locations where MOBs just go through with a non-zero speed, while stay points stand for locations where the MOB stays over a certain time interval.

Definition 2 (Stay point). *A stay point $\mathbb{L}_{sp}$ is a four tuple <**sID**, **x**, **y**, **Δt**>, where sID is the identifier of the stay point; <x, y> is the coordinate of the stay point, Δt is the duration of the stay.*

Each time a user stays at somewhere over a time interval, we can obtain a corresponding stay point. Stay points of a real-world place may have different coordinates. Suppose a person visits a shopping mall from the front gate, while another person visits the same mall from the back door. Although they visit the same shopping mall, the stay points we extract are different. On the other hand, the imprecision of GPS devices may also result in different stay points for a real-world place. In order to obtain the real-world places where the MOBs visit, we define the notion of *place*.

Definition 3 (Place). *A place $\mathbb{P}$ is a set of stay points, it is represented as <**pID**, **loc**, **add**, **sem**>, where pID and loc represent the identifier and the centroid coordinate of $\mathbb{P}$ respectively, add represents the address of $\mathbb{P}$, while sem represents the semantic characteristics of $\mathbb{P}$, which consists of three parameters $<\overrightarrow{v}$, Δt_{avg}, $t_{enter}>$, they represent the visitors, average visit duration and average enter time respectively.*

Places we define here correspond to real-world places, such as shopping malls, clubs, restaurants, etc. Each place is available for all MOBs to visit or stay.

Definition 4 (Zone). *A zone $\mathbb{Z}$ is an area consists of at least l places, it is represented as <**zID**, **bl**, **ur**, **pn**>, where zID is the identifier of the zone, bl and ur represent the coordinate of the bottom-left corner and the upper-right corner respectively, pn represents the number of places included by the zone.*

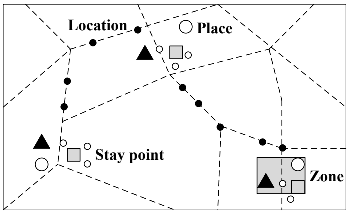


Fig. 2. Locations, stay points, places and zones

An example of these definitions can be seen in Fig.2, where solid back points are locations, hollow points represent stay points. Squares, circulares and triangles are real-world places, different shapes stands for different types of places. The shaded rectangle is a zone which contains three places. Zones are derived from places, places are derived from stay points. Thus, a zone is a generalized version of users' stay points.

4 Proposed Solutions

4.1 Solutions Overview

We assume adversaries have access to all published trajectories and the public background information. They know the distribution of real-world places on the map, but they do not know the movement parameters of MOBs. Before our method, we assume the traces are already anonymized by replacing the true identifier with a random and unique pseudonym. Our goal in this paper is to anonymize original trajectory database D to a published version D^*, in which stay points cannot be exposed in a probability larger than $1/l$. The procedure of *YCWA* is as follows (also shown in Fig.3):

- Split map generation. First, we extract stay points from raw trajectories, then reconstruct semantic places using a *reverse geocoder* [9]. After that, we construct zones containing l places through a grid-based and a clustering-based method respectively.
- Trajectory anonymization. We divide trajectories into {move, stay} sequences, where stay points are replaced by corresponding zones. Pass-by points are either deleted or un-processed, depending on wether it locates inside a zone or not. At last, D is transformed to D^* in this step.
- Information loss measure. We measure information loss of D^* in this step. Since D^* is always published for analysis purpose, the utility of D^* should be kept high. Here we adopt an information loss measure in [3], which is represented as the reduction of the probability with which people can accurately determine the position of a MOB.

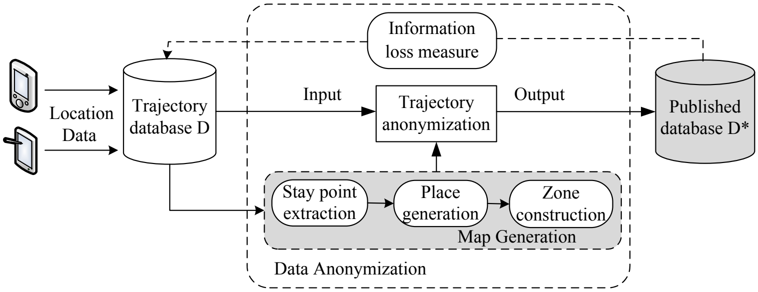


Fig. 3. Procedure of *YCWA*

4.2 Stay Points Extraction

We adopt stay points extraction strategies in [7] with improvements. Stay points are in two basic categories: stop and wondering. Accordingly, stay points occur in the following two situations. One of which is when a person equipped with a GPS logger gets into a building, the GPS logger loses signals and stops logging. Or, if a GPS-enabled car driver stops his car, the GPS device is turned off. The other one is when a GPS carrier is wondering around an outside sign, the GPS device is still logging and the velocity is not zero. In this case, the sign should also be regarded as a stay point [7].

For the first situation, we adopt a duration-based strategy. A parameter δt is introduced in order to avoid mistaking occasional stays for stay points, such as waiting for traffic lights. If a MOB stays at a location exceeding the time threshold δt, the location is regarded as a stay point. That is to say, given a trajectory $\mathbb{T}$={(L_1, t_1), (L_2, t_2), $\cdots$,

$(L_n, t_n)\}$, if $|t_{i+1}\text{-}t_i| > th_{time}$, L_i is regarded as a stay point, and the duration of this stay Δt is set to $|t_{i+1}\text{-}t_i|$. After that, all the stay points are put into D_{stays}. Thus, starts, ends and long stays are regarded as stay points.

For the second situation, we adopt a density-based strategy. Given a distance threshold th_{dist} and a time threshold th_{time}, if $distance(L_{i+1}, L_i) < th_{dist}$ and $|\ t_{i+1}\text{-}t_i\ | > th_{time}$, the MBR consists of L_i and L_{i+1} is called a dense area A_{dense}. Generally speaking, A_{dense} can be regarded as an outdoor stay point. However, a more complicated problem arises. When a GPS-enabled car meets traffic jams, the congestion area may be mistaken as an outdoor stay point. We solve this problem by recognizing whether the dense area A_{dense} is a road segment or not, since traffic jams always happen on road, while most outdoor signs are not. We put all the dense areas (represented by their geographic centers) as stay point candidates into D_{sc} and take D_{sc} into the next procedure.

4.3 Places Reconstruction

We recall Google Maps API to reverse-geocode coordinate of each stay point and stay point candidate. Places we reconstruct are put into D_{places}, which is initialized to empty. For each stay point L_i in D_{stays}, compare its reverse-geocoded address $L_i.add$ with each place′s address in D_{places}. If $L_i.add$ equals to $P_i.add$, merge L_i with P_i. Visitors $\overrightarrow{v}$, average visit duration Δt_{avg} and average enter time of the place t_{enter} of P_i are updated. While if $L_i.add$ does not equal to any addresses of existing places in D_{places}, set L_i as a new place in D_{places}. For each stay point candidate A_{dense} in D_{sc}, we reverse-geocode it at first. If the obtained address is a road segment, A_{dense} is probably caused by traffic jams, it should be deleted from D_{sc}. If the $A_{dense}.add$ is not a road segment, A_{dense} is regarded as an outdoor stay point, subsequent processing follows the same procedure as L_i. At last, we merge D_{sc} into D_{stay} and return D_{place}.

Real-world places are in different types, such as apartments, shopping malls, clubs, and office buildings,etc. In privacy-preserving literature, the most ideal situation is to enforce diversified places into a zone, but it is too hard to tag each place with a type. Therefore, we adopt a notion of similarity between places called place similarity [8] to solve this problem. We define similarity of places according to three parameters: *visitors*, *average visit duration* and the *average enter time*. Since places of different types usually exhibit different features. *e.g., office buildings may have an average enter time during 8.AM and 10.AM, average visit duration ranging from 7 to 9 hours, while a night club may be significantly different*. The three parameters can be used to capture these features and measure the similarities between places, as shown in equation (1).

Definition 5 (Place similarity). *Given two places* $< P_i$*, loc, add, sem*$>$ *and* $< P_j$*, loc, add, sem*$>$*, where sem is represented as* $< \overrightarrow{v}$, Δt_{avg}, $t_{enter} >$*. Place similarity can be computed by:*

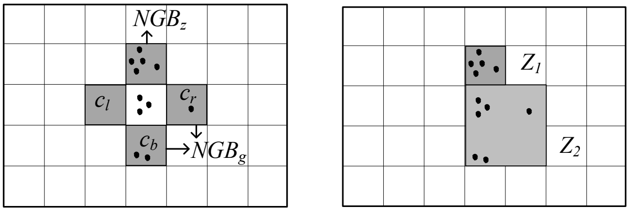
$$sim(P_i, P_j) = \frac{\overrightarrow{v_i} \cdot \overrightarrow{v_j}}{|\ \overrightarrow{v_i}\ ||\ \overrightarrow{v_j}\ |} + \frac{min(\Delta t_{avgi}, \Delta t_{avgj})}{max(\Delta t_{avgi}, \Delta t_{avgj})} + \frac{min(t_{enteri}, t_{enterj})}{max(t_{enteri}, t_{enterj})} \quad (1)$$

The vector space model is used to compute the similarity between two visitor lists. The similarity of average visit duration is computed as the smaller one divided by the larger one, the same is done for average enter time. Sim(P_i,P_j) is computed by linear combination of the three scores. The higher the sim(P_i, P_j) value, more similar they are.

4.4 Zones Construction

In this section, we turn places into zones in order to generate a split map. Two different approaches are proposed, one is grid-based approach, called GridPartition; the other one is clustering-based approach called DiverseClus.

GridPartition. In GridPartition, the whole 2D Euclidean space is uniformly divided into square cells. Each cell is called a grid. Obviously, places are located in different grids, and the number of places in each grid is different. $G_i.num$ denotes number of places contained by G_i. For a user specified privacy level l, not every grid contains enough places to be a zone. We design an enlarging strategy to enforce at least l places into a zone, as shown in Algorithm 1. Each grid G_i is scanned in a spatial order. If $G_i.num > l$, G_i is tagged as a zone, and put G_i into D_{zones} (line 3-6). If $0 < G_i.num < l$, we try to merge it with its *neighbors*. For any grid or zone G' near G_i, if $0 < G'.num < l$, it is regarded as G'_is grid neighbor, and then put it into NGB_g; while G'_is zone neighbors are put it into NGB_z (line 7-8). An example can be seen in Fig.4(a), where l is set to 5. When $G_i.num < l$, G_i enlarges itself by merging with its grid neighbors in NGB_g (line 10-12). If $NGB_g = \Phi$, G_i enlarges itself by merging with its zone neighbors in NGB_z (line 13-15). After merging, $G_i.num$, $G_i.ur$ and $G_i.bl$ should be updated, and G_i is put into D_{zones}. Fig.4(b) shows the resulted two zones using the enlarging strategy.



(a) (b)

Fig. 4. GridPartition

GridPartition guarantees each zone contains more than l places, but it doesn't take places' semantic meanings into consideration. If a zone contains l places of the same type, it may also lead to privacy leakage, as we have previously mentioned. DiverseClus is proposed to address this problem.

DivserseClus. Given two places P_i and P_j, based on the spatial distance and place similarity, we introduce a mixed distance measure defined in equation (2).

$$Dist_{mix}(P_i,P_j) = \frac{Dist(P_i,P_j)}{sim(P_i,P_j)+\alpha} \tag{2}$$

Algorithm 1: GridPartition (D_{places}, l)

```
Input  : $D_{places}$, minimum place numbers in each zone $l$
Output: $D_{zones}$
$D_{zones} \leftarrow \Phi$;
Divide the space into grids;
for each grid $G_i$ do
    if $G_i.num > l$ then
        $D_{zones} \leftarrow G_i$;
        continue;
    $NGB_g \leftarrow G_i$'s grid neighbors;
    $NGB_z \leftarrow G_i$'s zone neighbors;
    while $G_i.num < l$ do
        if $NGB_g \neq \Phi$ then
            randomly select a grid $g_i$ in $NGB_g$;
            merge $g_i$ into $G_i$;
            else
                randomly select a zone $z_i$ in $NGB_z$;
                merge $z_i$ into $G_i$;
        update $G_i.num$, $G_i.ur$ and $G_i.bl$;
    $D_{zones} \leftarrow G_i$;
return $D_{zones}$;
```

Here Dist(P_i, P_j) is a non-zero Euclidean distance, since P_i and P_j are represented by their geographic center, zero value never happens if they are two different places. The primary targets of DiverseClus is to cluster l places into zones with minimized area size, as well as diversified contents. We therefore measure the diversity of places by the dissimilarity of two places, represented as $\frac{1}{sim(P_1,P_2)+\alpha}$. The larger the value, more diverse they are, and they are more likely to be clustered together. α is used to avoid divide-by-zero error and smooth the penalty when the place similarity are very small. In our experiments, α is set as the standard deviation of place similarities, this strategy considers the majority differences of place similarities, and works well in our experiments. The details of the algorithm are represented in Algorithm 2.

At a very general level, the procedure of DiverseClus follows a similar structure of k-mediods [12]. The algorithm begins with a cluster center P_{cen}, which is the centroid place of D_{places}. Then, each cluster center is chosen as the farthest from the last one (except the first one, the *farthest* is measured by mixed distance, line 3-5). Places that near P_{cen} are clustered into $Clus$. $S_{P_{cen}}$ which represents clustering score of using P_{cen} as center is introduced to measure qualities of clusters. $S_{P_{cen}}$ is computed by the sum of distances of each place to P_{cen} in the cluster (line 6-8). In order to get an optimized result, the clusters should be adjusted by replacing P_{cen} with another place. Each place P_i in $Clus$ is selected to replace P_{cen}, the clustering score S_{Pi} is computed using P_i as the clustering center (line 9-11). If $S_{P_{cen}} < S_{P_i}$, replace P_{cen} with P_i, until no replacement happens (line 12-14). The clusters are represented by their minimum bounding rectangles (MBRs). At last, $Clus.num$, $Clus.bl$ and $Clus.ur$ are updated and $Clus$ is put into D_{zones}.

Algorithm 2: DiverseClus (D_{places}, l)

```
Input  : $D_{places}$, minimum place numbers required $l$
Output: $D_{zones}$
$P_{cen} \leftarrow$ the centroid place of $D_{places}$;
$D_{zones} \leftarrow \Phi$;
while $N_{cen} \leq \lfloor \, |D_{place}|/l \rfloor$ do
$P_{cen} \leftarrow$ the farthest of the last one;
$N_{cen}$++;
for each $P_{cen}$ do
    $Clus \leftarrow P_{cen} \cup l$-1 nearest neighbors of $P_{cen}$;
    $S_{P_{cen}} \leftarrow \sum_j \frac{Dist(P_{cen},P_j)}{sim(P_{cen},P_j)+\alpha}$;
    for each $P_i$ in $Clus$ do
        select $P_i$ to replace $P_{cen}$;
        $S_{P_i} \leftarrow \sum_j \frac{Dist(P_i,P_j)}{sim(P_i,P_j)+\alpha}$;
        if $S_{P_i} < S_{P_{cen}}$ then
            replace $P_i$ with $P_{cen}$;
            until no changes;
    update $Clus.num$, $Clus.ur$, $Clus.bl$;
    $D_{zones} \leftarrow Clus$;
return $D_{zones}$;
```

The clusters generated by DiverseClus should be post-processed, since some of the clusters may overlap spatially. This is because two places are dissimilar in semantic meanings while the spatial distance between them is far. We adjust the clusters using the following strategy. For each cluster *Clus* derived from DiverseClus, check if it spatially overlaps with other clusters. If so, merge the overlapped clusters until no overlap exists. After the merging, we need to check the place number *Clus.num* in each cluster. If $l < Clus.num < 2l$, *Clus* is a qualified cluster, and it should be put into D_{zones}. If $Clus.num > 2l$, split *Clus* into two non-overlapped clusters spatially, each cluster contains at least l places. After the adjustment, each cluster's MBR is regarded as a zone, a split map consists of zones is generated. It may be argued that, the adjustment may eliminate the effects of the place similarity, this is undesirable because most clusters are not overlapping or the overlapping region is relatively small, since the nearby places are very likely to be in different types, this can also be proved in our experiments.

4.5 Trajectory Anonymization

Trajectories are then split and anonymized based on the split map. The original trajectory database D is set as input, each location sample is scanned, stay points are replaced by the corresponding zones. For each pass-by point, the published version is kept as the original one, unless the pass-by point is *covered* by a zone. *Cover* is a spatial relationship between a zone and a pass-by point of the same trajectory, as defined in definition 6. If a pass-by point L_j is covered by a zone, L_j is suppressed for privacy-preserving purpose, since publication of location samples approaching to a zone may cause exposure of a stay point.

Definition 6 (Cover). *Given two location samples (L_i, t_i) and (L_j, t_j) on $\mathbb{T}$, L_i is a stay point, its corresponding zone is Z_i, while L_j is a pass-by point. If L_j locates inside Z_i, then L_j is* ***covered*** *by Z_i.*

5 Privacy and Utility Analysis

In this section we discuss the privacy guarantees and data utilities. We formally show that by applying our methods, the published database D^* will not expose any user's stay points during their travels. Privacy guarantee is always measured by re-identification probability which means the probability of adversaries to identify a stay point or a trajectory from the published database D^*.

Theorem 1. *Given a trajectory database D=$\{\mathbb{T}_1, \mathbb{T}_2, \ldots \mathbb{T}_n\}$ and its published version D^*=$\{\mathbb{T}^*_1, \mathbb{T}^*_2, \ldots \mathbb{T}^*_n\}$ generated by* YCWA*, the average stay points re-identification probability is bounded by 1/l.*

Proof. In the attack model, we assume adversaries have access to all the published trajectories and public knowledge. Adversaries do know the distribution of the places on the map, but they do not know the movement parameters of MOBs. Given a published version D^*, each stay point in D^* is generalized to an area which contains at least l diverse stay-able places. The re-identification probability depends on the number of places in a zone, which is bounded by $1/l$. □

To capture the information loss, we adopt the reduction in the probability with which people can accurately determine the position of an object in [3]. Given a published database D^* of D, the average information loss is defined in equation (3).

$$IL_{avg} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(1-1/area(zone(O_i,t_j))) + \sum_{d=1}^{h} L_d}{n \times m} \tag{3}$$

IL_{avg} represents the average shrinks of the identify probability of a location in D^*. Where $area(zone(O_i,t_j))$ represents the area size of the corresponding zone of O_i at time t_j when O_i stays. The probability of adversaries can accurately determine the location where the MOB stays shrinks from 1 to 1/area(zone(O_i, t_j)). If a location L_d is deleted, it is totally indistinguishable, so the information loss turns to be 1. $n \times m$ represents the total location samples in D. Obviously, IL_{avg} ranges from 0 to 1.

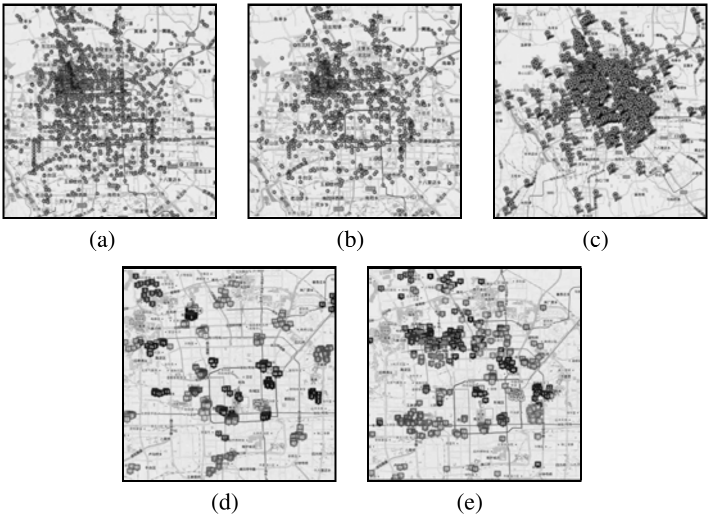
6 Experiments

In this section we report the empirical evaluation we have conducted in order to assess the performance of our methods, in terms of data utility and the efficiency.

6.1 Experimental Setup

We run our experiments on a real-world dataset. Thanks to the *Geolife* project [11], we get the published real trajectories of volunteers. The dataset contains more than 8000 trajectories of 155 users ranging from May 2007 to May 2010 mainly in Beijing. More than 23 million GPS records are contained. The dataset is represented as BEIJING henceforth. The experiments are run on an Intel Core 2 Quad 2.66HZ, windows 7 machine equipped with 4GB main memory.

Since we use the same dataset as in [7] to extract stay points, we adopt the same parameter values. Specifically, the duration threshold δ_t and th_{time} are set to 20 minutes, while the distance threshold th_{dist} is set to 200m. This results in 75,593 stay points (shown in Fig.5(b)) extracted from the BEIJING databaset (shown in Fig.5(a)). We may avoid bothering the readers with such details, as I believe we can simply clean the dataset first by removing all non-Beijing location points. It can be seen that, location samples distribute all over the city of Beijing, more than 95% of them concentrate within the Fifth Ring Road.


(a) (b) (c)

(d) (e)

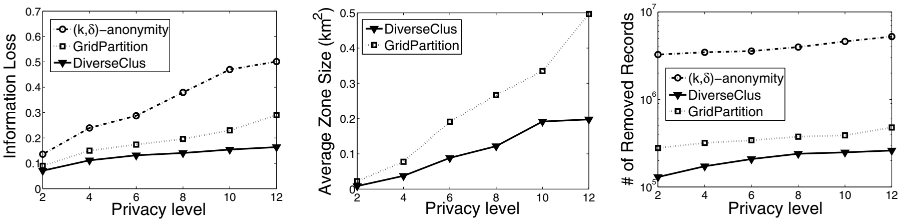
Fig. 5. Data distribution on the map. In (a) we report data distribution in BEIJING, in (b) stay points distribution, in (c) distribution of places, in (d) 50 clusters obtained purely on spatial distance, in (e) 50 clusters on mixed distance.

After extracting stay points in BEIJING, we recall Google Maps API to reverse each stay point to a real-world address. Thanks to Google Maps API, the returned results contain exact addresses and post codes, which make the place generation available and reasonable. After this procedure, 6902 semantic places are found. Fig.5(c) shows the distribution of semantic places in Beijing.

Both GridPartition and DiverseClus are used to generate the split map. In GridPartition, we divide the whole city of Beijing into grid cells of size 0.008° × 0.008° each, which results 62,408 grid cells. We then implement the enlarging strategies on these grids. In DiverseClus, the parameter α is set as the standard deviation of place similarities, as we have previously mentioned. The clustering results on spatial distance and mixed distance (without post-processing) are sampled in Fig.5(d) and 5(e), respectively. In both figures, 50 clusters are randomly selected to show the results. Places in the same cluster are painted in the same color. It can be seen that, clusters obtained based on mixed distance do overlap in Euclidean space representation before post-processing. Inclusion of place similarities do pose impacts on clustering results, post-processing is necessary in this situation. Based on the clustering results of places, the zones and the whole split map can be generated with various l values, i.e., the privacy levels. In the following experiments, we set l = 2, 4, 6 ,8, 10, and 12.

6.2 Measure of Data Utility

We then run a set of experiments on BEIJING to make a comparison on data quality between our approach and (k,δ)-anonymity [2]. (k,δ)-anonymity only works over trajectories with the same time span, a pre-processing step that partitions trajectories into equivalent classes is needed. Then a greedy clustering method is used to cluster trajectories together. At last, trajectories in each cluster are transformed into a (k,δ)-anonymity set, where δ is given as the radius of the anonymity set. The information loss we measure is computed by equation (3), where the area size is measured in square meters[1]. For (k, δ)-anonymity, the value of δ is set according to [2], ranges from 1000 to 4000, step by 1000. The evaluations shown in our figures are average values on δ. In information loss evaluation of (k, δ)-anonymity, we only account for the generalization and the deletion part, while the information loss caused by space translation can be seen in range query distortion measure.



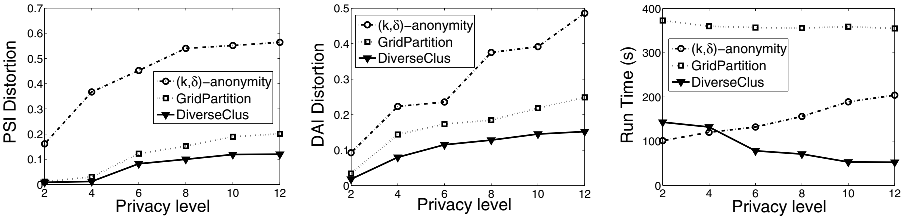
(a) (b) (c)

Fig. 6. Data utility measure of DiverseClus. In(a), we report information loss of the three algorithms, in (b) the average zone size, in (c) the number of removed location samples.

Comparison of all the three algorithms are shown in Fig.6(a). DiverseClus leads to less information loss than GridPartition since it adopts a clustering strategy, which

[1] The area size should be normalized by dividing 100, this is because the imprecision of GPS devices ranges from 5 to 15 meters, that is to say if a MOB locates in an approximately 100m^2 area, the location of the MOB can be identified.

makes the zone size smaller than GridPartition. Information loss of both DiverseClus and GridPartition are less than 20%, which obviously dominate (k,δ)-anonymity. The information loss caused by our proposal is mainly caused by generalization of stay points and deletion of *covered* pass-by points. We therefore measure the average zone size and number of deleted location samples in Fig.6(b) and Fig.6(c). Obviously, DiverseClus performs better than GridPartition on both metrics. Since smaller zone size may cover fewer pass-by points, thus, making the removed location samples reduced. In all three figures, performance decreases as *privacy level* grows. We do not measure anonymized region of (k, δ)-anonymity, since for a given δ, the anonymized region is fixed to $\pi(\frac{\delta}{2})^2$.



(a) (b) (c)

Fig. 7. Performance evaluation of 3 algorithms, in (a) we report *PSI* query distortion comparison, in (b) the comparison of *DAI* distortion, in (c) run time comparison of 3 algorithms

We then measure the actual distortion of range query results on the published dataset D^* from the original dataset D. In particular, given a spatial region R and a time duration $[t_s, t_e]$, we consider two range queries the same as [2]: *Possibily_Sometimes_Inside* and *Definitely_Always_Inside*, represented as *PSI* and *DAI* for short respectively. Range query distortion is measured by $Distor_{rq} = \frac{min(Q(D),Q(D_*))}{max(Q(D),Q(D_*))}$. Where *Q(D)* represents the number of query results on D, $Q(D^*)$ stands for the number of query results on D^*. We measure query distortion for various l values. We randomly choose circular region R having radius between 500 and 5000, and randomly choose time interval ranging from 2 hours to 8 hours. The parameter settings we use are according to [2]. At last, 1000 queries are generated, each of them have 1000 runs. We run these queries on trajectory database anonymized by our approaches and (k, δ)-anonymity. The average query distortion is shown in Fig.7(a) and Fig.7(b). Both GridPartition and DiverseClus have a range query distortion under 20%, while for (k,δ)-anonymity, the query distortion is larger, almost up to 60%, since it adopts space translation, some location samples are translated to totally different ones, making the query results lost.

6.3 Measure of Efficiency

The running times evaluation of the three algorithms are shown in Fig.7(c). Running time of GridPartition is the longest of the three, since the partition into grids procedure is costly. In both GridPartition and DiverseClus, we exclude the time consumption of stay points extraction and place reconstruction, because Google Maps API contains restrictions, it is only allowed to reverse one coordinate every 2 seconds. The procedure of stay points extraction and places generation cost about 2951 minutes, but the data set of places can be used on various *l* values for both methods.

7 Conclusions

Collection and publication of people's everyday trajectories pose serious threats on people's personal privacy. In this paper, we propose to protect trajectory privacy through protecting significant stays on their trajectories, which can avoid unnecessary anonymization of pass-by location samples. Although sometimes the privacy guarantee of *YCWA* is not better than trajectory *k*-anonymity, in some applications, *YCWA* works well and the information loss is much lower.

In the future, we plan to reinforce our approach for multiple attack models as well as improve the space similarity measure. In *YCWA*, we assume adversaries do not know the moving speed of a MOB. But if they know the moving speed, such as, by knowing the maximal moving speed of a MOB, adversaries may infer the reachability to a zone, thus, the re-identification probability may increase. In another aspect, we plan to extend our approach to an online scenario, which means to dynamically maintain the split map when a new comer enters an area, thus, it can protect people's real-time trajectory privacy efficiently.

Acknowledgments. This research was partially supported by the grants from the Natural Science Foundation of China (No. 61070055, 60833005, 91024032, 91124001), the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University of China (No. 11XNL010, 10XNI018), National Science and Technology Major Project (No. 2010ZX01042-002-003).

References

1. Nergiz, M.E., Atzori, M., Saygin, Y., Baris, G.: Towards Trajectory Anonymization: A Generalization-based Approach. IEEE Transactions on Data Privacy 2, 47–75 (2009)
2. Abul, O., Bonchi, F., Nanni, M.: Never Walk Alone: Uncertainty for Anonymity in Moving Objects Databases. In: 24th IEEE International Conference on Data Engineering, pp. 215–226. IEEE Press, Washington (2008)
3. Yarovoy, R., Bonchi, F., Lakshmanan, S., Wang, W.H.: Anonymizing Moving Objects: How to Hide a MOB in a Crowd? In: 12th International Conference on Extending Database Technology, pp. 72–83. ACM Press, New York (2009)
4. Terrovitis, M., Mamoulis, N.: Privacy Preservation in the Publication of Trajectories. In: 9th International Conference on Mobile Data Management, pp. 65–72. IEEE Press, Washington (2008)
5. Gidofalvi, G., Huang, X., Bach, P.T.: Privacy-preserving Trajectory Collection. In: 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, p. 46. ACM Press, New York (2008)
6. Moreale, A., Andrienko, G.L., Andrienko, N.V., Giannotti, F., Pedreschi, D., Rinzivillo, S., Wrobel, S.: Movement Data Anonymity through Generalization. IEEE Transactions on Data Privacy 3, 91–121 (2010)

7. Zheng, Y., Zhang, L., Xie, X., Ma, W.: Mining Interesting Locations and Travel Sequences from GPS Trajectories. In: 18th International Conference on World Wide Web, pp. 791–800. ACM Press, New York (2009)
8. Cao, X., Cong, G., Jensen, C.S.: Mining Significant Semantic Locations From GPS Data. Proceedings of the VLDB Endowment 3(1), 1009–1020 (2010)
9. Google Reverse Geo-coder, http://www.findlatitudeandlongitude.com/find-address-from-latitude-and-longitude.php
10. Gruteser, M., Liu, X.: Protecting Privacy in Continuous Location-tracking Applications. IEEE Security and Privacy 2(2), 28–34 (2004)
11. Microsoft Research Geolife, http://research.microsoft.com/en-us/projects/geolife/
12. Kaufmann, L., Rousseeuw, P.: Clustering by means of medoids. Statistical Data Analysis Based on the L1-Norm and Related Methods, 405–416 (1987)
13. Xu, T., Cai, Y.: Exploring Historical Location Data for Anonymity Preservation in Location-based Services. In: 27th Conference on Computer Communications, pp. 547–555. IEEE Press, Washington (2008)

Semi-Edge Anonymity: Graph Publication when the Protection Algorithm Is Available

Mingxuan Yuan and Lei Chen

Department of Computer Science and Engineering
The Hong Kong University of Science & Technology
{csyuan,leichen}@cse.ust.hk

Abstract. With the popularity of social networks, the privacy issues related with social network data become more and more important. The connection information between users, as well as their sensitive attributes, should be protected. There are some proposals studying how to publish a privacy preserving graph. However, when the algorithm which generates the published graph is known by the attacker, the current protection models may still leak some connection information. In this paper, we propose a new protection model, "Semi-Edge Anonymity", to protect both user's sensitive attributes and connection information even when an attacker knows the publication algorithm. Moreover, any state-of-art tabular data protection techniques can be applied to Semi-Edge Anonymity model to protect sensitive attributes. We theoretically prove that on two utilities, the possible world size and the true edge ratio, the Semi-Edge Anonymity model outperforms any clustering based model which protects links. We further conduct extensive experiments on real data sets for two other utilities. The results show that our model also has better performance on these utilities than the clustering based models.

1 Introduction

Social network websites are becoming more and more popular in recent years. A social network graph contains lots of information, such as users' age, name, education background and relationship between users, etc. Fig. 1 is a social network example, where each user is represented as a node and the relationship between a pair of users is represented as a link. When publishing such a graph, it is important to protect the privacy of the involving users [1][9][12]. For many applications, a social graph is modeled as a labeled graph, which contains the following information [2][3][6][20]:

- Node Information:
    - The non-sensitive attributes of a user. Similar to the tabular micro-data, we call them quasi-identifiers. E.g., the education background and age on each node in Fig. 1 are the quasi-identifiers[1].
    - The sensitive attributes of a user. E.g., in Fig. 1, the salary of each user is the sensitive attribute. We also call the sensitive attributes the sensitive labels.
- Link information: the relationships between users. We also call them the structure information.

[1] Note that, the letters in vertices are not vertex labels, they are vertex IDs that we introduce to simplify description.

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 367–381, 2012.
© Springer-Verlag Berlin Heidelberg 2012

When publishing a graph, the link information as well as the sensitive attributes should be protected [2][6]. The attack using certain background knowledge to query the published graph to find a node/link is called 'node/link re-identification'. We call the protection of the link information the *link protection* and the protection of the sensitive attributes the *node protection*. The two basic methods for graph publication are edge-editing (i.e. to add or delete edges to make the graph satisfy certain properties) and clustering (i.e. to divide nodes into clusters and publish a clustered graph. The nodes/edges in a clustered graph are also called super nodes/edges.). There are two models [2][3][6] that can have the quantifiable guarantee on both the node protection and the link protection[2] when an attacker knows arbitrary subgraphs around victims. These models well preserved the user privacy as they were proposed. However, they may still suffer privacy leakage when the attacker combines the graph generation policy.

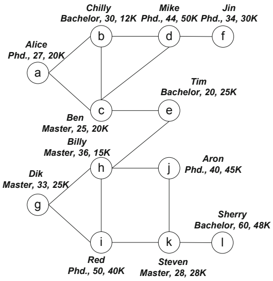
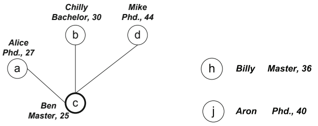
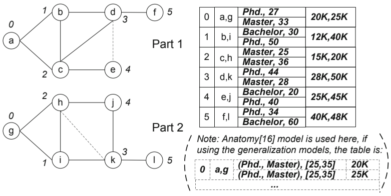


Fig. 1. Social network example



(a) Ben's subgraph 1 (b) Billy and Aron

Fig. 2. Possible background knowledge of an attacker

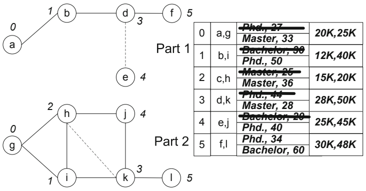


0	a,g	Phd., 27 Master, 33	20K,25K
1	b,i	Bachelor, 30 Phd., 50	12K,40K
2	c,h	Master, 25 Master, 36	15K,20K
3	d,k	Phd., 44 Master, 28	28K,50K
4	e,j	Bachelor, 20 Phd., 40	25K,45K
5	f,l	Phd., 34 Bachelor, 60	40K,48K

Note: Anatomy[16] model is used here, if using the generalization models, the table is:

0	a,g	(Phd., Master), [25,35] (Phd., Master), [25,35]	20K 25K
...			

Fig. 3. 2-Isomorphism Graph



0	a,g	~~Phd., 27~~ Master, 33	20K,25K
1	b,i	~~Bachelor, 30~~ Phd., 50	12K,40K
2	c,h	~~Master, 25~~ Master, 36	15K,20K
3	d,k	~~Phd., 44~~ Master, 28	28K,50K
4	e,j	~~Bachelor, 20~~ Phd., 40	25K,45K
5	f,l	Phd., 34 Bachelor, 60	30K,48K

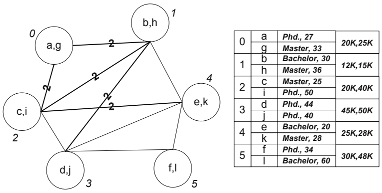
Fig. 4. Attack Demonstration

The first model [6], "k-isomorphism model", is based on edge-editing for both node protection and link protection. The protection objective is: for an attacker with arbitrary subgraph knowledge, the probability of discovering any user's information or finding any two users have any connection is at most $\frac{1}{k}$. A graph is k-isomorphism if this graph consists at least k disjoint isomorphic subgraphs. Fig. 3 is a 2-isomorphism graph of Fig. 1. Since there are at least k disjoint isomorphic subgraphs, each node is "hidden" with as least $k-1$ other nodes. For link protection, since any node's candidates appear

[2] The probability of re-identifying a node and the probability of learning any two nodes have a link is bounded by a constant.

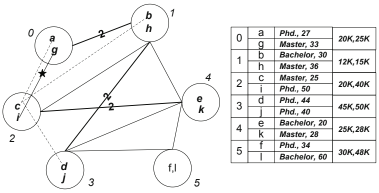
in at least k disjoint isomorphic subgraphs, the probability an attacker finds two nodes have a connection is at most $\frac{1}{k}$. For example, the candidates of Tim are $\{e, j\}$ and the candidates of Ben are $\{c, h\}$. So, the mappings between candidates and users (Tim, Ben) are $\{(e, c), (e, h), (j, c), (j, h)\}$. Due to disjoint subgraphs, at most two of them contain an edge between the two mapping nodes. Therefore, the probability that an attacker finds two nodes have a link is at most $\frac{1}{2}$.

However, it is required to make as fewer edge changes as possible to keep the graph utility [6]. The published graph is generated only with a small portion of edge changes[3] [6]. Therefore, the probability that two connected nodes are put into different disjoint parts is low. Suppose a published graph is generated with $p\%$ edges changed. If an attacker knows a connected subgraph G' (e.g. Fig. 2(a)), for any two nodes in G', the probability these two nodes are put into two different subgraphs is at most $p\%$, since it is required to delete at least one edge in order to put these two nodes into two separated subgraphs[4]. When an attacker knows a subgraph as shown in Fig. 2(a), he can find the mapping between candidates and users, for example, {Alice, Chilly, Mike} should either be {a,b,d} in part 1 or be {g,i,k} in part 2 of Fig. 3 with very high confidence (at least 1-$p\%$ where p is small). Thus the edges such as e(Alice, Chilly) and e(Chilly, Mike) are released. In some cases, even when an attacker does not know that two nodes are in the same subgraph, the link between these two nodes can still be discovered. For example, if an attacker knows Ben's subgraph (as shown in Fig. 2(a)) and the quasi-identifiers of Billy and Aron (shown in Fig. 2(b)), then he knows Billy and Aron are not in Fig. 2(a). After that, he can delete the subgraph about Ben (Fig. 2(a)) from Fig. 3. Although he might not have any idea about Fig. 2(a) appears in which isomorphism part of Fig. 3, he can still randomly select one and remove Fig. 2(a). The new graph is shown in Fig. 4, which is not a 2-isomorphism graph. The link protection, such as the link between Billy and Aron (i.e. link between h and j), cannot be guaranteed anymore.



0	a	Phd., 27	20K,25K
	g	Master, 33	
1	b	Bachelor, 30	12K,15K
	h	Master, 36	
2	c	Master, 25	20K,40K
	i	Phd., 50	
3	d	Phd., 44	45K,50K
	j	Phd., 40	
4	e	Bachelor, 20	25K,28K
	k	Master, 28	
5	f	Phd., 34	30K,48K
	l	Bachelor, 60	

Fig. 5. A 2-anonymous clustered graph



0	a	Phd., 27	20K,25K
	g	Master, 33	
1	b	Bachelor, 30	12K,15K
	h	Master, 36	
2	c	Master, 25	20K,40K
	i	Phd., 50	
3	d	Phd., 44	45K,50K
	j	Phd., 40	
4	e	Bachelor, 20	25K,28K
	k	Master, 28	
5	f	Phd., 34	30K,48K
	l	Bachelor, 60	

Fig. 6. A 2-anonymous clustered graph under Attack

Different from the first model, Cormode [2][3] proposed a clustering based model which also protects both the nodes and the links. For an attacker with arbitrary subgraph

[3] At most around 10% of edges are changed in [6]'s experiments.

[4] Actually, to make the mappings $\{(e, c), (e, h), (j, c), (j, h)\}$ have the equal probability of occurrence, at least 50% edges in the original graph should be changed. For any two nodes u and v where there exists one edge (u, v), to let an attacker believe u and v only have $\frac{1}{k}$ probability to be assigned into the same connected subgraph, (u, v) should be deleted with probability $\frac{k-1}{k}$. Even with $k = 2$, at least 50% edges should be changed. This violates the graph generation policy of k-isomorphism model.

knowledge, their model guarantees the probability of node re-identification and link re-identification is at most $\frac{1}{k}$. By making each cluster's size at least k, the probability of re-identifying a user can be bounded to be at most $\frac{1}{k}$. For link protection, Cormode [2][3]'s protection model requires the following two safety conditions: (1) No links within a cluster; (2) For any two nodes in the same cluster, they do not connect to the same node. For the graph in Fig. 1, a clustered graph with $k = 2$ as shown in Fig. 5 can be published[5]. For any two super nodes C_x and C_y, if we use $|C_x|$ and $|C_y|$ to represent the number of nodes contained in C_x and C_y, this clustering model constrains that the number of edges between C_x and C_y is at most $min\{|C_x|, |C_y|\}$. Since each cluster's size is at least k, the probability that an attacker can find any two nodes have a link is at most $\frac{min\{|C_x|,|C_y|\}}{|C_x||C_y|} \leq \frac{1}{k}$.

However, the above safety conditions might lead privacy leakage. Consider the following case: If an attacker knows a subgraph as shown in Fig. 2(a), he can uniquely find Dik and Red has a link by combining the published graph with the two safety conditions (shown in Fig. 6). There are two edges between super nodes 0 and 2. Since there already exists an edge $e(a, c)$ based on the background knowledge (i.e. Fig. 2(a)) and the clustering condition makes sure these two edges do not connect to the same node, the left edge must be between (g, i) (Dik and Red). This attack works for homogeneous graphs (there are at most one edge between any two nodes) since the safety condition provides more information than the published graph.

From the above analysis, we can see there indeed exists a problem when an attacker knows the generation policy of the published graph. To solve this problem, in this paper, by assuming an attacker knows our graph generation policy as well as arbitrary subgraphs:

- We give a necessary and sufficient condition for clustering based models, which can guarantee the link protection objective by restricting the number of edges between and within clusters.
- We propose a new protection model named Semi-Edge Anonymity (*SEA*) model, which separately protects nodes and edges. We give a safety random algorithm to implement this model. The new model performs well in the preservation of original graph utilities.

The SEA model can plug in any state-of-art protection model for tabular data to protect sensitive labels. Although we use a random algorithm to generate the published graph, the SEA model well preserves the graph's structure information. We prove that the SEA model outperforms any clustering based model which protects links on two utilities. Extensive experiments on real data sets also show that our SEA model well preserves two other utilities too.

2 Problem Description

In this paper, we develop a graph protection model for both node protection and link protection. Our model guarantees the following two protection objectives:

[5] The number on each super edge is the number of edges this super edge represents. If a super edge only represents one edge, we don't display the number "1" to make the Figures clean.

Objective 1 Any node u, the probability that an attacker can re-identify u's sensitive label is at most $\frac{1}{k}$;

Objective 2 Any nodes u and v, the probability that an attacker can successfully find u and v have a link is at most $\frac{1}{k}$. We use $Prob(con(u,v))$ to denote this probability. Then this objective is $Prob(con(u,v)) \leq \frac{1}{k}$.

where k is a pre-given constant.

We suppose an attacker has the following background knowledge:

- The quasi-identifiers of victims;
- Any labeled subgraphs around victims[6];
- The method used to generate the published graph;

An attacker could use the above information to query the published graph to learn user's sensitive labels or to find whether two users have a link in the published data.

3 Safety Condition for Clustering

For any clustering based method, to guarantee protection objective 1 (see in our problem definition), a necessary condition is that each cluster's size is at least k. To guarantee the link protection objective (protection objective 2), the following necessary and sufficient *Safety Clustering Condition* must be satisfied:

1. Any super node C, the number of edges d_C between the nodes within C must satisfy $d_C \leq \frac{|C|(|C|-1)}{2k}$;
2. Any two super nodes C_x and C_y, the number of edges d_{C_x,C_y} between the nodes in C_x and the nodes in C_y must satisfy: $d_{C_x,C_y} \leq \frac{|C_x||C_y|}{k}$;

Proof.
- Sufficient: Any nodes u and v,:
  - If $u \in C \wedge v \in C$: $Prob(con(u,v)) = \frac{d_C}{\binom{|C|}{2}} = \frac{2d_C}{|C|(|C|-1)} \leq \frac{2\frac{|C|(|C|-1)}{2k}}{|C|(|C|-1)} = \frac{1}{k}$
  - If $u \in C_x \wedge v \in C_y$: $Prob(con(u,v)) = \frac{d_{C_x,C_y}}{|C_x||C_y|} \leq \frac{\frac{|C_x||C_y|}{k}}{|C_x||C_y|} = \frac{1}{k}$

  In both cases, protection objective 2 is guaranteed.
- Necessary
  - If there exists a super node C, $d_C > \frac{|C|(|C|-1)}{2k}$, then $\forall u,v \in C$:

$$Prob(con(u,v)) = \frac{d_C}{\binom{|C|}{2}} = \frac{2d_C}{|C|(|C|-1)} > \frac{2\frac{|C|(|C|-1)}{2k}}{|C|(|C|-1)} = \frac{1}{k}$$

    Protection objective 2 is violated.
  - If there exist any two super nodes C_x and C_y, $d_{C_x,C_y} > \frac{|C_x||C_y|}{k}$, then $\forall u \in C_x \wedge \forall v \in C_y$

$$Prob(con(u,v)) = \frac{d_{C_x,C_y}}{|C_x||C_y|} > \frac{\frac{|C_x||C_y|}{k}}{|C_x||C_y|} = \frac{1}{k}$$

    Protection objective 2 is violated.

We further use this necessary and sufficient condition to analyze the utilities of any clustering based models when link protection must be provided in Section 6.

[6] The quasi-identifier of a node u is actually involved in any labeled subgraph around u. To be clear, we represent them separately here.

4 Semi-Edge Anonymity Model

Properly combining the Safety Clustering Condition with a clustering based model can achieve the link protection (protection objective 2). However, the current edge-editing based models and clustering based models both mix the node protection and link protection together. The link protection is implemented based on node grouping, which brings unnecessarily structure information loss. For example, in clustering based models, the edge is anonymized based on clusters. A demonstration example is shown in Fig. 7(a). If there's only one edge between clusters C_x and C_y, the number of possible positions to put this edge is $|C_x||C_y|$, which is at least k^2. However, to achieve protection objective 2, we only need k positions to allocate an edge. Thus, if we can directly anonymize this edge, the number of positions to put this edge can be reduced significantly, better preserving the structure information. In this paper, we propose the idea of Semi-Edge to anonymize each edge directly.

Definition 1. *Semi-Edge: a Semi-Edge $se(u, U)$ is a pair where u is a node and U is a set of nodes. $\forall v \in U$, we say se covers edge $e(u, v)$.*

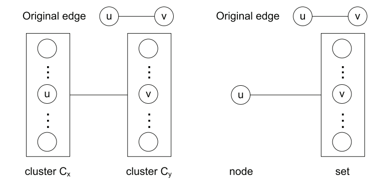
We use $|U|$ to denote the size of node set U. From one Semi-Edge $se(u, U)$ where $(|U| = k)$, an attacker can only have $\frac{1}{|U|} = \frac{1}{k}$ probability to find u has a link with any node in U. An example of Semi-Edge is shown in Fig. 7(b). If we use a Semi-Edge to anonymize the edge between C_x and C_y, the number of possible positions to put it is reduced to k (for theoretical analysis, please check Section 6).

It has been shown that the node protection can only be guaranteed when the grouping of nodes considers the sensitive attributes' diversity and distribution (e.g. different l-diversity models[13], t-closeness [11] etc.) as well as the cost function to generate the published data (minimality attack [15]). Current graph protection models either implement k-anonymity [2][4][6][12][19][21][17][9] or l-diversity [20][5]. They all rely on a specific protection model. If we separate the node protection and link protection, the protection of sensitive labels are not restricted to a specific protection model. Motivated by this, instead of using the edge-editing based model and clustering model, we proposed a new graph protection model, Semi-Edge Anonymity (SEA) model, to protect node and link information independently.

The SEA model publishes two tables separately. The first table, attribute table, contains only quasi-identifiers and sensitive attributes following any state-of-art tabular data protection model[13][11][15]. The second table, Semi-Edge table, anonymizes each edge directly by using a Semi-Edge. A simple example is shown in Fig. 8. There are two benefits of our SEA model:

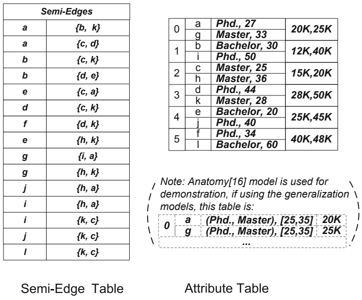
- Any state-of-art sensitive label protection models for tabular data can be directly adopted to the SEA model. The generation of attribute table and Semi-Edge table can be done independently;
- Directly anonymizing each link is helpful to maintain the graph utility.

In the rest part of this paper, we use $Q(u)$ to represent node u's quasi-identifiers and $S(u)$ to represent u's sensitive attributes. To protect the node information, we partition nodes into groups and publish the quasi-identifiers/sensitive attributes of nodes based



(a) Clustering-based Model (b) Semi-Edge

Fig. 7. Motivation example for utility improvement



Semi-Edges	
a	{b, k}
a	{c, d}
b	{c, k}
b	{d, e}
e	{c, a}
d	{c, k}
f	{d, k}
e	{h, k}
g	{i, a}
g	{h, k}
j	{h, a}
i	{h, a}
i	{k, c}
j	{k, c}
l	{k, c}

Semi-Edge Table

0	a	Phd., 27	20K,25K
	g	Master, 33	
1	b	Bachelor, 30	12K,40K
	i	Phd., 50	
2	c	Master, 25	15K,20K
	h	Master, 36	
3	d	Phd., 44	28K,50K
	k	Master, 28	
4	e	Bachelor, 20	25K,45K
	j	Phd., 40	
5	f	Phd., 34	40K,48K
	l	Bachelor, 60	

Note: Anatomy[16] model is used for demonstration, if using the generalization models, this table is:

0	a	(Phd., Master), [25,35]	20K
	g	(Phd., Master), [25,35]	25K
...			

Attribute Table

Fig. 8. A Semi-Edge Graph

on these groups in an attribute table. Attribute table can reuse any existing protection model for sensitive labels in tabular data such as l-diversity[13], t-closeness[11] etc. We give each node an anonymity id in order to link with the Semi-Edge table. To be simple, in the rest part of this paper, we use u to directly represent the id of u. For example, if we choose the anatomy model defined in [16], v_1, ..., v_m form a group, then a tuple $(\{(v_1, Q(v_1)), ..., (v_m, Q(v_m))\}, \{S(v_1), ..., S(v_m)\})$ is published for this group as shown in Fig. 8[7]. The graph we publish is:

Definition 2. *Semi-Edge Graph: A Semi-Edge Graph SG(*AT*,* SET*) represents* $G(V, E)$ *by the pair* AT *and* SET*.* AT *is an attribute table for all the nodes in* G*.* SET *is a Semi-Edge table. All edges in* E *are covered by the Semi-Edges in* SET*.*

For example, Fig. 8 is a Semi-Edge Graph of Fig. 1. Since any node in SET is mapped to a group of sensitive labels in AT, it is obvious the protection of sensitive labels and links are independent:

Lemma 1. *In a Semi-Edge Graph SG, the node protection in* AT *is not influenced by* SET *and the link protection in* SET *is not influenced by* AT*, either.*

AT and SET can be generated independently. There are lots of existing works on how to generate an attribute table, in this paper, we focus on how to implement the link protection in SET.

We give a *Safety Semi-Edge Condition* which can guarantee that a Semi-Edge Graph SG achieves protection objective 2:

Con 1 $\forall se(u, U) \in SET, |U| = k$;
Con 2 $\forall se(u, U) \wedge \forall v \in U, (\forall se(u, U'), v \notin U') \wedge (\forall se(v, U'), u \notin U')$;

There are two constraints in our safety condition. The first constraint requires that each Semi-Edge covers k different edges. The second constraint requires that for any edge $e(u, v)$ covered by one Semi-Edge, no other Semi-Edges, which can cover it, appear in SET. For example, in Fig. 8, edge $e(a, b)$ and $e(a, k)$ are covered by Semi-Edge

[7] If we use the models, which generalize all the quasi-identifiers in each group to be the same (i.e. $Q(v_1)...Q(v_m)$ are generalized to $GQ_{1...m}$), a tuple $(\{v_1, ..., v_m\}, \{(GQ_{1...m}, S(v_1)), ..., (GQ_{1...m}, S(v_m))\})$ is published.

$(a, \{b, k\})$, and none of other Semi-Edges covers $e(a, b)$ or $e(a, k)$. A Semi-Edge graph SG which satisfies the Safety Semi-Edge Condition is called *Semi-Edge anonymized graph*. Our SEA model is to publish a Semi-Edge anonymized graph instead of the original graph.

Lemma 2. *A Semi-Edge anonymized graph always guarantees the protection objective 2 when an attacker knows the published graph is generated under the Safety Semi-Edge Condition.*

Proof. Firstly, the two constraints in the Safety Semi-Edge Condition could be directly observed in the published data. This guarantees that no extra information is released according to the safety condition. Any node u and v, suppose the attacker uses the strongest background knowledge (i.e. the whole labeled graph without $e(u, v)$) to query the published graph. In $SG(AT, SET)$, there at most exists one Semi-Edge $se(u, U)$ or $se(v, U)$ such that se covers $e(u, v)$. According to Con 1 in the Safety Semi-Edge Condition, any $|U| = k$. So for any two nodes u and v, the probability that an attacker finds they have a link is at most $\frac{1}{k}$.

5 Generation Algorithm

Algorithm 1: Generate Semi-Edge Table

```
1  set $SET = \{\}$ ;
2  Hashtable $ht$ = new Hashtable();
3  for each $e(u, v)$ in $E$ do
4      Semi-Edge $se = new(u, U = \{v\})$ ;
5      while $|U| < k$ do
6          Randomly select node $v'$ where $v' \neq u \wedge v' \notin U$ ;
7          $U = U \cup \{v'\}$ ;
8      for each $v'$ in $U$ do
9          if $ht.contains(e(u, v'))$ then
10             $ht[e(u, v')] = ht[e(u, v')] + 1$;
11         else
12             $ht$.add($e(u, v')$,1);
13 while true do
14     $e(u, v)$ is the edge with the maximum value in $ht$;
15     if $ht[e(u, v)] > 1$ then
16         Select $se(u', U')$ where $\exists v' \in U', e(u', v') = e(u, v)$ and $se(u', U')$ is not created due to $e(u, v)$;
17         Randomly select a node $w$ with $e(u', w) \notin ht \wedge w \neq u' \wedge w \notin U'$;
18         $U' = U' - \{v'\} + \{w\}$;
19         $ht[e(u, v)] = ht[e(u, v)] - 1$;
20         $ht$.add($e(u', w)$,1);
21     else
22         break;
```

We use Algorithm 1 to generate the Semi-Edge table SET. For each original edge $e(u, v)$ in G, we generate a Semi-Edge $se(u, U)$. We firstly add v into U, which promises se cover $e(u, v)$. Then we randomly add $k - 1$ other nodes into U to make sure each Semi-Edge covers k edges (lines 4 - 7). We use a hash table ht to record the number of Semi-Edges that covers each possible edge (lines 8 - 12). If each element in ht has value 1, the Safety Semi-Edge Condition is satisfied. If there are some edges covered by more than one Semi-Edge, we use the following method to adjust SET to make it satisfy the Safety Semi-Edge Condition:

1. We find the edge $e(u, v)$ which is covered by the maximum number of Semi-Edges in SET ($e(u, v)$ is the edge with the maximum value in ht).

2. If $e(u, v)$ is only covered by one Semi-Edge, the Safety Semi-Edge Condition is satisfied, we finish our adjustment (lines 20-21).
3. Otherwise, we find a Semi-Edge $se(u', U')$ which covers $e(u, v)$ where $se(u', U')$ is not originally created due to edge $e(u, v)$[8]. We randomly select one node w where $e(u', w) \notin ht$ and replace v' with w. By doing this, the number of Semi-Edges that cover $e(u, v)$ is decreased by 1.

We recursively do the above steps until SET satisfies the Safety Semi-Edge Condition.

Theorem 1. *A Semi-Edge anonymized graph always guarantees the protection objective 2 when an attacker knows the published graph is generated by Algorithm 1.*

Proof. According to Lemma 2, knowing the Safety Semi-Edge Condition does not cause the privacy leakage. In Algorithm 1, the nodes in each Semi-Edge are randomly selected, therefore the algorithm does not guarantee any certain output when given an input. Given a SET, any graph which is consistent with SET can be the input of this algorithm. Thus, a Semi-Edge anonymized graph generated by Algorithm 1 always guarantees the protection objective 2.

6 Utility Analysis

We use a random algorithm to guarantee no privacy leakage during the generation process. In this section, we compare our model with clustering based models by analyzing two utilities, the possible world size and the true edge ratio (see definitions in later paragraphs). We show although we enhance the privacy by using a random algorithm, our model always performs better than or equal to any clustering based models which consider link protection on these two utilities.

For clustering based models, since only super nodes and super edges are published, each published graph represents a group of graphs which are consistent with it. The set of graphs $W(G)$ that are consistent with the published graph is named as the possible world [9][10]. When using such a clustered graph, users can sample some graphs in $W(G)$ and compute the average values of these graphs. So, the number of consistent graphs of a clustered graph $|W(G)|$ should be as less as possible [9][10]. We call $|W(G)|$ the possible world size. When generating a clustered graph, the one with smaller $|W(G)|$ is preferred. So, firstly, we analyze the number of graphs that are consistent with the published graph ($|W(G)|$).

In a clustering based model, $|W(G)|$ is computed as [9][10]:

$$|W(G)|_{clustering} = \prod_{\forall C} \binom{\frac{|C|(|C|-1)}{2}}{d_C} \prod_{\forall C_x, C_y} \binom{|C_x||C_y|}{d_{C_x, C_y}}$$

Where d_C is the number of edges between the nodes in cluster C and d_{C_x, C_y} is the number of edges between the nodes in cluster C_x and C_y.

[8] Since the Semi-Edge covering $e(u, v)$ could be $se(u, U_1)$ or $se(v, U_2)$, we use u' here to avoid the confusion in the description. We use $v' \in U'$ to repsent the other endpoint of $e(u.v)$.

In a Semi-Edge anonymized Graph, any Semi-Edge $se(u,U)$, the original edge covered by se can be between u and any node in U, $|W(G)|$ is computed as:

$$|W(G)|_{SEA} = \prod_{\forall se(u,U)} |U| = k^{|E|}$$

Theorem 2. *A Semi-Edge anonymized graph always has $|W(G)|$ which is smaller than or at most equals to any clustered graph when link protection should be provided.*

Proof. In a clustered graph: $|W(G)|_{clustering} = \prod_{\forall C} \binom{\frac{|C|(|C|-1)}{2}}{d_C} \prod_{\forall C_x,C_y} \binom{|C_x||C_y|}{d_{C_x,C_y}}$.

For any two integer numbers M and x with $1 \leq x \leq \frac{M}{k}$, $\binom{M}{x} = \frac{M(M-1)...(M-x+1)}{x(x-1)...1}$. For any $y \in [0, x-1]$, $\frac{M-y}{x-y} \geq \frac{M-y}{\frac{M}{k}-y} = k + \frac{yk^2-yk}{M-yk} \geq k$. So if $1 \leq x \leq \frac{M}{k}$, $\binom{M}{x} \geq k^x$.

We proved in Section 3, the Safety Clustering Condition is a necessary and sufficient condition for any clustered graph to provide link protection. The Safety Clustering Condition requires:

- Any super node C, the number of edges d_C between nodes within C must satisfy $d_C \leq \frac{|C|(|C|-1)}{2k}$;
- Any two super nodes C_x and C_y, the number of edges d_{C_x,C_y} between the nodes in C_x and the nodes in C_y must satisfy: $d_{C_x,C_y} \leq \frac{|C_x||C_y|}{k}$;

So $\binom{\frac{|C|(|C|-1)}{2}}{d_C} \geq k^{d_C}$ and $\binom{|C_x||C_y|}{d_{C_x,C_y}} \geq k^{d_{C_x,C_y}}$. We can get:

$$\begin{aligned} |W(G)|_{clustering} &\geq \prod_{\forall C} k^{d_C} \prod_{\forall C_x,C_y} k^{d_{C_x,C_y}} \\ &= k^{\sum_{\forall C} d_C} \times k^{\sum_{\forall C_x,C_y} d_{C_x,C_y}} = k^{|E|} = |W(G)|_{SEA} \end{aligned}$$

Next we show for another utility, our Semi-Edge anonymized graph also works better than or at least equals to any clustered graphs. Suppose G' is a sampled graph from the published graph. If an edge appears both in G' and the original graph G, we call this edge a true edge. We use the ratio of true edges in G' to estimate how much structure information of G is correctly represented by G'. The ratio of true edges TR is defined as: $TR = \frac{\text{no. of true edges in } G'}{|E|}$. The larger TR is, the better G' represents G. We use $expTR$ to represent the expected ratio of true edges in any sampled graph.

Theorem 3. *If using uniform random sampling method, the $expTR$ of any sampled graph from a Semi-Edge anonymized graph is always larger than or at least equals to the $expTR$ of any sampled graph from a clustered graph when link protection should be provided.*

This theorem can be proved similar to Theorem 2.

7 Experiments

The analysis in the model description part proves the privacy protection effectiveness of our model. As the same as other privacy preserving graph publication works

[12][19][21][6][17][9][18][4][8][2], we test several utilities to show how well the published data preserves the structure information of the original graph. We test three real data sets and compare our SEA model with two clustering based models. The three real data sets we tested are: Cora (www.cs.umd.edu/projects/linqs/projects/lbc/index.html, 2708 nodes and 5429 edges), Arnet (www.arnetminer.net/, 6000 nodes and 37848 edges), and ArXiv (arXiv.org, 19835 nodes and 40221 edges). We only compare with clustering based models since the edge-editing based model with link protection can be regarded as a sampled graph of the SEA model. For the edge-editing based model, to make it protect links, we should let the published graph contain at most $\frac{1}{k}|E|$ randomly selected true edges. Such a published graph is one sampled graph of the Semi-Edge anonymized graph. However, if we only publish one sampled graph, the information in it is biased since the information of deleted edges ($\frac{k-1}{k}|E|$) are totally missed.

7.1 Utilities

We have proven the SEA model outperforms any clustering based model on $|W(G)|$ and $expTR$. For a published graph that represents more than one graph, to test how good the published graph is, researchers sample a group of graphs that are consistent with the published graph, calculate graph proprieties in these sampled graph and use the average property changes as graph change benchmarks [9][18][4][8][2]. In our experiment, besides $|W(G)|$ and $expTR$, we also sample n ($n = 100$) graphs to compute the average graph property changes. Suppose the sampling graph set is S, we test the change ratio of the following two graph properties:

- Average change ratio of degrees (ACR_D)

$$ACR_D = \frac{\sum_{\forall G_s \in S} \frac{\sum_{\forall u} \frac{|degree(u)_{G_s} - degree(u)_G|}{degree(u)_{G_s} + degree(u)_G}}{|V|}}{n}$$

- Average change ratio of clustering coefficient (ACR_{CC})
  The CC of a vertex in a graph is commonly used[12][21] to represent its neighborhood graph. It is defined as the actual number of edges between the vertex's directed neighbors divides the max possible number of edges between these directed neighbors. We test the average change ratio of the clustering coefficients:

$$ACR_{CC} = \frac{\sum_{\forall G_s \in S} \frac{\sum_{\forall u} \frac{|CC(u)_{G_s} - CC(u)_G|}{CC(u)_{G_s} + CC(u)_G}}{|V|}}{n}$$

Smaller $|W(G)|$, ACR_D, ACR_{CC} and larger $expTR$ are preferred.

7.2 Clustering-Based Models

We compare our SEA model with two clustering based models.

- D-Clustering (Directly-Clustering)[9][10]
  Hay *et al.* use Simulated Annealing (SA) algorithm to generate clustered graph which prevents node re-identification[9][10]. The only limitation to generate the clustered graph is that each cluster's size should be at least k. The SA algorithm targets to minimize the $|W(G)|$. There are three operations in SA: (1) Merge two

clusters to one cluster; (2) Split one cluster to two clusters; (3) Move nodes between two clusters. It showed that after running SA algorithm for around $100|V|$ steps, the $|W(G)|$ becomes stable. We would like to use this model as the baseline. In our experiment, we set the stop condition of SA algorithm as running at least $120|V|$ steps and in the last 1000 steps, no better solution was found.

- S-Clustering (Safely-Clustering)
  We enhance the clustering based model by adding the Safety Clustering Condition when generating the clusters. We also use the SA algorithm to find a solution with as smaller $|W(G)|$ as possible. The only difference is besides the cluster size constraint, the number of edges within any cluster and between any two clusters is further bounded by the Safety Clustering Condition. The stop condition is set as the same as D-Clustering.

7.3 Results

Fig. 9 shows the results of $|W(G)|$ on the three data sets. From the results we can see, comparing with the S-Clustering which provides link protection, our SEA model performs much better. In all cases, $|WG|_{S-Clustering} > |WG|^2_{SEA}$. Even for D-Clustering which does not have link protection and directly optimizes the $|W(G)|$, the SEA model has much better performance in nearly all cases (except the four points in ArXiv graph when $k \leq 5$).

We demonstrate the results of $expTR$ in Fig. 10. In all cases, the SEA model publishes a graph with higher $expTR$ than the S-Clustering. In most cases, the SEA model also outperforms D-Clustering (except the five points in ArXiv graph when $k \leq 6$), but D-clustering does not offer any link protection.

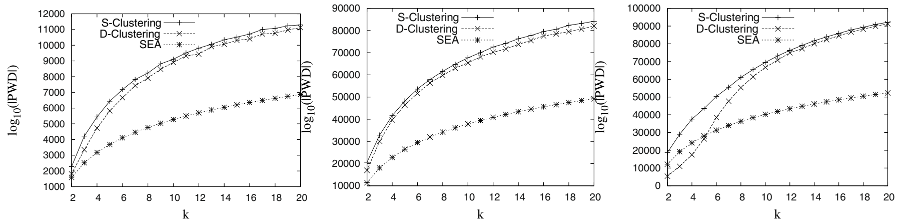
Fig. 11 represents the results of ACR_D. For data sets Cora and Arnet, the SEA model has smaller ACR_D than both S-Clustering and D-Clustering for all ks. For data set ArXiv, the SEA model always performs better than S-Clustering. For $k \in [16, 20]$, the SEA model has similar performance as D-Clustering. For other cases ($k \in [2, 15]$), the SEA model also has smaller ACR_D than D-Clustering.

Fig. 12 represents the results of ACR_{CC}. For all data sets, the SEA model performs better than S-Clustering and D-Clustering.

From the results, we can see the SEA model has better performance than S-Clustering on the four utilities for all data sets. For most cases, the SEA model even has better performance than D-Clustering, which does not consider link protection. The experiment results confirm the effectiveness of structure information preserving by directly anonymizing each edge. Although our SEA model uses a random generation algorithm to guarantee the privacy, we can still preserve the graph utilities well.

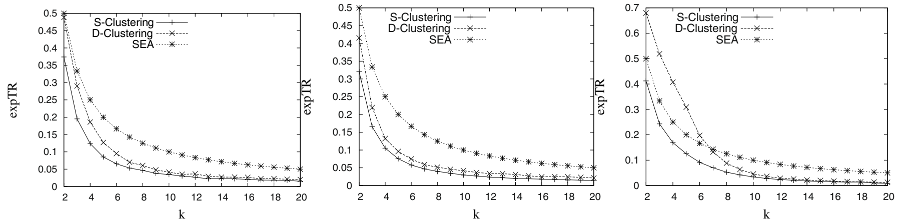
8 Related Works

There are many works having been proposed to address privacy issues on publishing social network graphs. The attack that uses certain background knowledge to re-identify the nodes/links in the published graph is called "passive attack". There are two models



(a) Cora (b) Arnet (c) ArXiv

Fig. 9. The results of $|W(G)|$



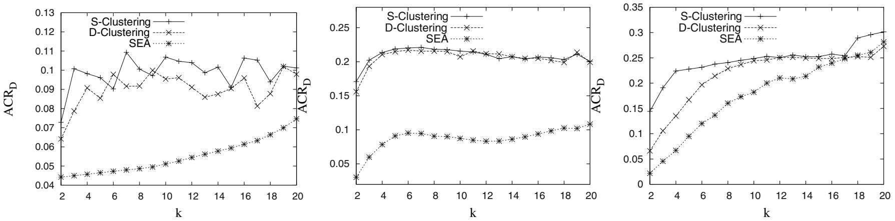
(a) Cora (b) Arnet (c) ArXiv

Fig. 10. The results of $expTR$

proposed to publish a privacy preserving graph against the passive attack: edge-editing based model [12][19][21][6][17] and clustering based model [9][18][4][8][2].

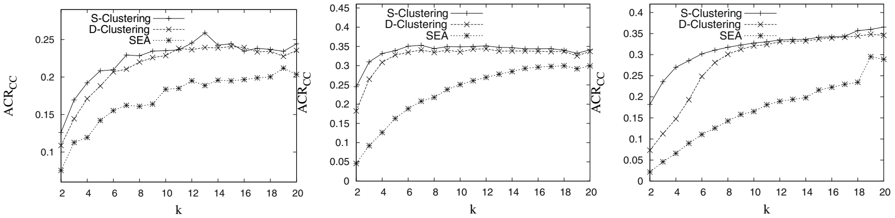
Most graph protection models implement k-anonymity [14] of nodes on different background knowledge of the attacker. Liu[12] defined and implemented k-degree-anonymous model, that is in a published graph, for any node, there exists at least $k-1$ other node having the same degree as this node. Zhou[19] considered a stricter model: for every node, there exist at least $k-1$ other nodes sharing isomorphic neighborhoods when taking node labels into account. In paper [20], the k-neighborhood anonymity model is extended to l-neighborhood-diversity model to protect the sensitive node labels. Probability l-diversity is implemented in this model. Zou[21] proposed a k-Automorphism protection model: A graph is k-Automorphism if and only if for every node there exist at leasts $k-1$ other node without having any structure difference with it. Hay[9] proposed a heuristic clustering algorithm to prevent node re-identification. Campan[4] discussed how to implement a clustering model against subgraph attack to nodes when considering both node labels and structure information lost. Cormode[8][2] introduced (k,l)-groupings for bipartite graph against attacks using subgraphs. Compan [5] implemented a p-sensitive-k-anonymity clustering model which requires each cluster satisfy k-anonymity and distinct l-diversity.

Besides the protection of nodes, several works considered the protection of link information. Zheleva[18] developed a clustering method to prevent the sensitive link leakage. Ying[17] studied how random deleting and swapping edges change graph properties and proposed an eigenvalues oriented random graph changing algorithm. These works did not provide a quantifiable guarantee on link protection [6]. Cheng[6] designed a k-isomorphism model to protect both nodes and links: a graph is k-isomorphism if this graph consists at least k disjoint isomorphic subgraphs. The attributes of nodes are



(a) Cora (b) Arnet (c) ArXiv

Fig. 11. The results of ACR_D



(a) Cora (b) Arnet (c) ArXiv

Fig. 12. The results of ACR_{CC}

protected by anatomy model [16] in a k-isomorphism graph. The k-isomorphism graph guarantees that an attacker at most have $\frac{1}{k}$ probability to find two nodes have a connection in case he knows arbitrary subgraphs. Cormode[2][3] introduced a clustering based model which implements node protection through k-anonymity and makes sure that an attacker at most have $\frac{1}{k}$ probability to find two nodes have a connection in case he knows arbitrary subgraphs. However, when an attacker has the information of published graph generation policy, the link protection cannot be guaranteed in some cases.

Our SEA model handles passive attack for both node protection and link protection even when an attacker knows our graph generation policy as well as arbitrary subgraphs. We do not restrict the node protection to any specific models such as k-anonymity [14] or k-anatomy [16]. Any state-of-art tabular data protection models such as different l-diversity models [13], t-closeness model[11] etc. can be adopted into our model. Our SEA model directly anonymizes each edge, which helps to reduce the structure information loss in the published graph.

9 Conclusion

In this paper, we propose a new protection model, Semi-Edge Anonymity, to solve the problem when the graph publishing algorithm is known to the attacker. The model could provide link protection and node protection at the same time. An algorithm is proposed with randomness for the SEA model. We prove the new model can protect links well even when the attacker knows the generation algorithm. The other important benefit of the SEA model is that it can adopt any state-of-art sensitive label protection models for tabular data. This makes the SEA model also have the ability to provide the best protection to sensitive labels. We theoretically prove the SEA model outperforms any clustering based models to provide link protection on two utilities. Experiment shows the SEA model also performs better than clustering based models when link protection is considered on two other utilities. In our future work, we will consider stronger attackers who know some knowledge about the relationship between non-sensitive attributes and links.

Acknowledgments. This work is supported in part by Hong Kong RGC grants N_HKU-ST612/09, National Grand Fundamental Research 973 Program of China under Grant 2012CB316200 and NSFC 619308.

References

1. Backstrom, L., et al.: Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In: WWW (2007)
2. Cormode, G., et al.: Class-based graph anonymization for social network data. In: VLDB (2009)
3. Bhagat, S., et al.: Prediction promotes privacy in dynamic social networks. In: WOSN (2010)
4. Campan, A., et al.: A clustering approach for data and structural anonymity in social networks. In: PinKDD (2008)
5. Campan, A., et al.: P-sensitive k-anonymity with generalization constraints. In: TDP (2010)
6. Cheng, J., et al.: K-Isomorphism: Privacy Preserving Network Publication against Structural Attacks. In: SIGMOD 2010 (2010)
7. Cormode, G., et al.: Anonymizing bipartite graph data using safe groupings. In: PVLDB (2008)
8. Cormode, G., et al.: Anonymizing bipartite graph data using safe groupings. In: PVLDB (2008)
9. Hay, M., et al.: Resisting structural re-identification in anonymized social networks. In: PVLDB (2008)
10. Hay, M., et al.: Resisting structural re-identification in anonymized social networks. In: VLDBJ (2010)
11. Li, N., et al.: t-closeness: Privacy beyond k-anonymity and l-diversity. In: ICDE (2007)
12. Liu, K., et al.: Towards identity anonymization on graphs. In: SIGMOD (2008)
13. Machanavajjhala, A., et al.: L-diversity: Privacy beyond k-anonymity. In: TKDD (2007)
14. Sweeney, L., et al.: k-anonymity: a model for protecting privacy. In: FKBS (2002)
15. Wong, R., et al.: Minimality attack in privacy preserving data publishing. In: VLDB (2007)
16. Xiao, X., et al.: Anatomy: Simple and effective privacy preservation. In: VLDB (2006)
17. Ying, X., et al.: Randomizing social networks: a spectrum preserving approach. In: SDM (2008)
18. Zheleva, E., Getoor, L.: Preserving the Privacy of Sensitive Relationships in Graph Data. In: Bonchi, F., Malin, B., Saygın, Y. (eds.) PInKDD 2007. LNCS, vol. 4890, pp. 153–171. Springer, Heidelberg (2008)
19. Zhou, B., et al.: Preserving privacy in social networks against neighborhood attacks. In: ICDE (2008)
20. Zhou, B., et al.: The k-anonymity and l-diversity approaches for privacy preservation in social networks against neighborhood attacks. In: KAIS (2010)
21. Zou, L., et al.: K-Automorphism: A General Framework For Privacy Preserving Network Publication. In: VLDB (2009)

On-the-Fly Generation of Facets as Navigation Signs for Web Objects

Yu Kawano, Hiroaki Ohshima, and Katsumi Tanaka

Department of Social Informatics,
Graduate School of Informatics, Kyoto University
{kawano,ohshima,tanaka}@dl.kuis.kyoto-u.ac.jp

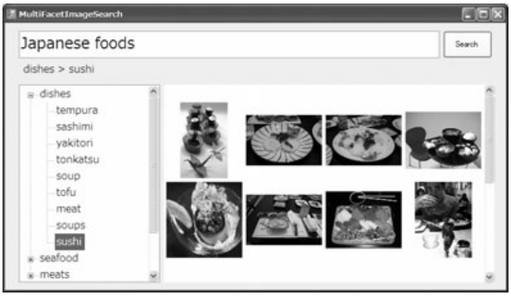
Abstract. We propose a method to generate facets dynamically to enhance the navigation of objects returned by a web-based search query. Facets denote axes for classifying a currently viewed object and related objects and are used as navigation signs to indicate their positions. Facets are generated by detecting hypernyms and coordinate terms of expressions that characterize objects. To be effectively used for browsing search results, generated facets are ranked. We implemented a prototype system that shows returned images from an image search classified by multiple facets. The results of an experiment to assess the facets showed that the average precision of correct facets in all queries obtained using our system is up to 82.7% for the top three and up to 77.6% for the top five ranked facets.

1 Introduction

Increasingly, search engines offer effective retrieval functions for searching several types of web objects, such as web pages, images, and videos. In most search engines, users input keywords as queries. However, keyword-based search systems are somewhat problematic. First, it is difficult for average users to create precise queries that adequately reflect their information needs. This is particularly the case when a user is unfamiliar with targeted fields, e.g., a user who wants images of Japanese "sushi" but does not know the specific names of types of sushi. Even if usersf information needs are clear, they cannot create a precise query [2,17]. Second, relationships between searched objects are not presented. In general, a search result is shown in a listed form. When a user browses for an object, moving to other related objects is difficult.

A faceted interface is expected to be effective for solving these problems. A facet denotes an axis for classification. It is a powerful tool for navigating and refining search results because it presents appropriate facets depending on the targeted items. For example, images of "Japanese foods" are classified by facets of "dishes" and "ingredients" that are specific to Japanese foods. In this example, if a faceted interface is introduced, users can browse images of "sushi" by selecting the category "sushi" from the facet "dishes." In addition, when a user browses images of "sushi," he/she can move directly to lists of images of "sashimi" and "tempura," which are other Japanese food "dishes." In this manner, facets allow users to refine and navigate results by clarifying the position of objects in the search results.

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 382–396, 2012.
© Springer-Verlag Berlin Heidelberg 2012



Japanese foods

dishes > sushi

Fig. 1. Screenshot of the proposed system

Similarly, facets can support the transition of keyword queries by positioning targets for searches on the Web. For example, when a keyword query is "Kyoto $\wedge$ Japanese foods," the facet "cities" enables a user to move directly to a search result for the query "Tokyo $\wedge$ Japanese foods."

In this paper, we propose a method for generating facets "on-the-fly" as navigation signs for objects returned by keyword searches. The prototype for the proposed faceted interface system is shown in Fig. 1. The system generates facets by rapidly finding hypernyms, coordinate terms, and hyponyms. The generated facets are evaluated to experimentally determine whether they are suitable for browsing search results. In addition, the effectiveness of the facet interface is validated through a user study.

Our prototype system was evaluated on its ability to dynamically generate "single-word"-based facets. Whenever a user selects a web object among search results, the system generates the most closely related facets, each of which consists of a single word. In this paper, we also propose a method of generating "multiple-words"-based facets.

Our contributions are summarized as follows.

- We proposed a method of dynamically generating related facets for a user-selected web object by using the indices of a web search engine. This approach is different from the selection of given facets in advance.
- We proposed a method of selecting effective facets for browsing a search result.
- We built a system by which effective facets for browsing a search result were presented and evaluated the utility of the system.
- We proposed an improved method of generating "multiple-words"-based facets.

2 Related Work

The use of facets has been reported to be effective for navigating and refining searches [6]. A faceted search system is mainly applied to objects in a specific domain, such as product searches and music searches. In these systems, the candidates for facets are almost always provided in advance. Many studies have been conducted in which appropriate facets are selected from a given set of facets [16,3,1,18,12]. In contrast, other studies report faceted interfaces that are automatically generated from dictionaries and large-scale databases [8,9]. In the area of facet selection, Tunkelang [16] proposed a method of determining dynamic category sets of multiple facets that best match the query, and Fumas et al. considered the vocabulary problem in human-system communication [4]. Li et al. [8] proposed a method for generating a query-dependent faceted interface for Wikipedia articles.

In this paper, techniques for extracting hyponyms and pairs of a hypernym and coordinate terms are used. Coordinate terms have a hypernym in common. Thus, our approach is closely related to techniques for extracting terms of a specific relationship. In this area, studies for extracting coordinate terms are active [5,14]. Shinzato and Torisawa [14] acquired coordinate terms from HTML documents. Furthermore, other studies have reported the extraction of terms from various relationships by changing lexico-syntactic patterns [11].

3 Facets

3.1 Definition of Facet and Object Sets for Classification

A facet denotes an axis for classifying an object set. For example, photos can be classified by axes of "color," "size," and "month shot."

In this study, we have established some requirements for facets. (1) A facet is set of categories, and it is desirable for facets to be orthogonal. (2) Categories in an individual facet are flat. (3) An object is classified into at least one category.

Orthogonal facets are independent. Ideally, facets should be orthogonal but some are not. For example, in a classification of photos, the nonorthogonal facets of "month shot" and "season shot" could both be considered. In general, categories of facets are hierarchical or flat. For example, the granularity of "place" varies (e.g., country, state, and city), and therefore, "place" could be composed of hierarchical categories. In facets with many categories, hierarchical categories are desirable. A facet of "color" could be composed of flat categories, such as "red" and "blue." In this paper, facets are determined to be flat. An object may be classified into multiple categories. In Fig. 2, two photographs with the facet "football player" are classified into three categories: "Beckham," "Zidane," and "Ronaldinho." As illustrated in the right-hand photo, multiple items can be pictured; thus an object is not necessarily classified into one category. Hence, in this paper, an object can be classified into multiple categories.

A facet and a facet set are defined as follows in this paper:

Definition 1. A facet F_i is a set of categories denoted by $F_i = \{f_1^i, f_2^i, \ldots, f_k^i\}$, where f_k^i is a category. An object can be classified into multiple categories.
Definition 2. A facet set is denoted by $\mathcal{F} = \{F_1, F_2, \ldots, F_l\}$.

Our goal is to classify an object set returned by a keyword-based search engine by facets. The object set is defined as described below:

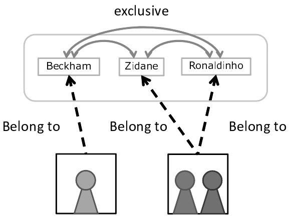


Fig. 2. Example of classifying images of "football player" by the "name" facet

Definition 3. Given a keyword query q, an object set returned by a search engine is denoted by $O = \{o_1, o_2, \ldots, o_m\}$, where o_m is an object.

In this study, an "*effective*" facet set $\mathcal{F}$ is dynamically generated to classify O. In the next two sections, we describe dynamic facet generation and "*effective*" facets, repsectively.

3.2 Dynamic Facet Generation

In Bing Images, faceted interfaces where facets for "size" and "color" of images are presented are provided. By contrast, an image search result returned by the query "Japanese foods" are classified by facets of "dishes" and "ingredients" that are specific Japanese foods. Facets that are dependent of a classified object set are denoted by dynamic facets.

Many studies where appropriate facets are dynamically selected from given facets are conducted [16,3,1,18,12]. Yee et al. [18] proposed a method of classifying given images by facets that are a part of the hierarchical structure in WordNet [10]. In this study, facets are not known in advance but are generated dynamically. Without dictionary-based knowledge, facets are generated on-the-fly depending on the object set. Note that our purpose is dynamic facet generation as opposed to dynamic facet selection, which has been the focus of many previous studies.

3.3 Effective Facets

In this section, we describe "*effective*" facets. Two types of measurements pertain to an "*effective*" facet: independent of other facets and dependent on other facets. Measurements that are independent of other facets are listed as follows.

Relevance of a Facet to an Object Set: An "*effective*" facet must be relevant to an object set. For example, images of a "soccer player" cannot be classified by the facet "manufacturer," which is inappropriate for the images.

Uniformity in Granularity of Categories: In an "*effective*" facet, uniform granularity of categories is desirable. For example, suppose that the facet "place" is composed of the categories "Rome," "France," and "England." Here, the categories that represent both country and city are included, and therefore, the granularity of categories is not uniform.

Uniformity in Size of Categories: In an "*effective*" facet, the number of objects should be uniform in all categories.

A facet that fulfills these requirements is defined as an "*effective*" facet.

Orthogonality between facets measures whether an individual facet is "*effective*." For example, facets of "color" and "size" are orthogonal because they are independent of each other. In this paper, a set of "effective" facets that considers orthogonality between facets is called an effective facet group.

3.4 Problem Definition

We attempt to solve two problems in this paper.

Problem 1. Generate a facet set $\mathcal{F}$, which is a set of "*effective*" facets, for classifying an object set O returned from a search engine by the keyword query q.

Problem 2. Generate an effective facet group from $\mathcal{F}$.

In this paper, we do not consider the problem of facet selection and focus intensively on Problem 1.

We will elaborate on generating facets to classify O in Section 4 and present calculations for the effectiveness of facets in Section 5.

3.5 Presentation of Facets

Our goal is to generate a set of "*effective*" facets and classify an object set in a search result by the facets. Users who refine their search in the faceted interface can select a facet and its categories and browse objects classified into the categories. For ease of use, the names of facets and their categories are denoted as "facet name" and "category name," respectively.

A facet name and a category name have two relationships: "is-a" and "part-of." In an "is-a" relationship, a facet name represents a super concept of a category name. In a "part-of" relationship, a category name is part of a concept represented by a facet name. In this paper, we focus on facets containing the "is-a" relationship.

4 Generating a Facet Set

In this section, we describe a method to generate a facet F_i to classify an object set O. Facets where the relationship between a facet name and a category name is an "is-a" relationship are generated. The process of generating facets is listed as follows.

1. **Extraction of Candidates for Category Names:** A set of terms that characterize the objects T is extracted. T is composed of nouns, noun phrases, and candidates for category names.

2. **Detection of a Facet Name:** Pairs of a hypernym and a set of coordinate terms are constructed using $t \in T$. A detected hypernym and coordinate terms correspond to a facet name and category names, respectively.
3. **Validation of Facets:** The candidates for facets are evaluated by detecting hyponyms of facet names.

Facet generation has two primary phases: detection of a facet name and validation of facets.

4.1 Finding Facet Names

We describe a method of generating candidates for facets by finding pairs of a hypernym and coordinate terms of $t \in T$. The context of the coordinate terms presents an important problem associated with the pairs. For example, coordinate terms of "apple" has two different contexts. When the context is "company," "Microsoft" and "Google" are coordinate terms of "Apple." However, when the context is "fruit," "orange" and "banana" are coordinate terms of "apple." If these two types of coordinate terms are combined, the granularity of categories will not be uniform.

We propose a method to detect a hypernym and coordinate terms in parallel, considering the granularity and the context of coordinate terms. We now focus on the lexico-syntactical pattern "such as" for uniformity in granularity of coordinate terms [7]. The process of finding pairs of a hypernym and coordinate terms is given below.

1. Using a web search engine, a query ⟨"such as t"⟩ is executed. "A" recognizes A as a phrase.
2. From titles and snippets in the returned result, sentences that include the pattern "such as t" are extracted.
3. A hypernym and coordinate terms are detected from each sentence, and a pair comprising them is generated.
4. The pairs with a common hypernym are merged into one pair.

As an example, we explain this process when t is "the Eiffel Tower." First, in steps 1 and 2, sentences in which the pattern "such as the Eiffel Tower" occurs are collected. In the phrase "tourist spots such as the Eiffel Tower and the Louvre Museum," "tourist spots" is the hypernym and "the Louvre Museum" is the coordinate term of "the Eiffel Tower." In step 3, a pair of a hyponym and coordinate terms is extracted from the phrase. The coordinate terms are in the same context because a hypernym and coordinate terms are detected in one phrase. In step 4, pairs that have a common hypernym are merged and are replaced by a new pair composed of a common hypernym and the merged coordinate terms. The pairs correspond to candidates for facets. For example, the pair of the hyponym "tourist spots" and the set of coordinate terms and t, ("the Eiffel Tower" and "the Louvre Museum") corresponds to a facet candidate. Through the process, candidates for facets $\mathcal{F}_t$ are generated.

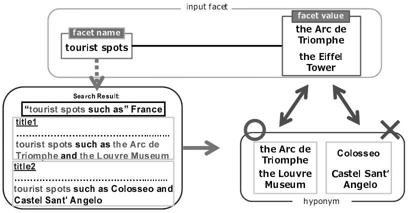


Fig. 3. Example of the facet validation process

4.2 Validation of Facets

In this section, we describe how facets are validated by detecting hyponyms of their facet names. If a facet F_i is correctly generated by finding hypernyms and coordinate terms, it is expected that some of the hyponyms of the facet name of F_i are included in category names of F_i. The process for the validation of F_i is listed as follows.

1. The facet name of F_i is denoted by n_i. The query $\langle$"n_i such as" $\wedge$ $q\rangle$ input into a web search engine, where q is the keyword.
2. From titles and snippets in the returned results, sentences that include the pattern "n_i such as" are extracted.
3. A hyponym set of n_i is detected in each sentence.
4. If the hyponym set and the categories of F_i intersect, F_i is regarded as valid. In this case, the terms in the hyponym set are added to categories of F_i.
5. The processes in steps 3 and 4 are performed on all sentences. If F_i is not valid in all sentences, then it is regarded as inadequate.

Fig. 3 illustrates the process for validating a facet when q is "France." With regard to step 1, it is important to modify queries for hyponym detection using q. If the query $\langle$"tourist spots such as"$\rangle$ is entered without "France," "Kyoto" and "the Great Wall" are likely to be obtained as hyponyms of "tourist spots," which are irrelevant and undesirable for the validation of F_i. In the extraction of hyponyms from sentences, we focus on descriptions that occur after "tourist spots such as". Given the phrase "tourist spots such as the Arc de Triomphe and the Louvre Museum," " the Arc de Triomphe" and "the Louvre Museum" are hyponyms of " tourist spots." In step 4, F_i is validated by comparing the hyponyms to category names of F_i. Both the hyponyms and the category names include "the Arc de Triomphe." This implies that "tourist spots" is a super concept of "the Arc de Triomphe" and "the Eiffel Tower." Therefore, F_i is regarded as valid. At the same time, the concepts of the hyponyms and the category names are thought to have uniform granularity. Thus, all the hyponyms are added to the set of category names. As a result, the category names of F_i are obtained as "the Arc de Triomphe," "the Eiffel Tower," and "the Louvre Museum."

In this manner, a valid F_i is obtained, and all facets included in $\mathcal{F}_t$ are validated. The above process is performed for all $t \in T$ and multiple valid facets are generated.

5 Scoring Facets

In this section, we describe the process of scoring facets based on measurements of "*effective*" facets.

Here, relevance of a facet to an object set is expressed by the co-occurrence of the facet name n_i and the keyword query q. $\mathrm{Rel}(n_i, q)$ denotes the page count for the query $\langle n_i \wedge q \rangle$. It is expected that $\mathrm{Rel}(n_i, q)$ is higher if n_i is closely related to q.

A facet name plays a part in the uniformity of the granularity of categories. If it is not clear whether a facet name is a super concept of a category name, then uniformity in granularity of categories $\mathrm{Granularity}(F_i)$ is not guaranteed, which is expressed as follows:

$$\mathrm{Granularity}(F_i) = \mathrm{freq}(n_i) \cdot \sum_{c \in C_i} \mathrm{cooccur}(n_i, c)$$

where C_i denotes the set of category names of F_i, $\mathrm{freq}(n_i)$ represents the occurrence of the pattern "n_i such as" through the generation and validation of F_i, $\mathrm{cooccur}(n_i, c)$ represents the co-occurrence of n_i and $c \in C_i$ through the generation and validation of F_i. When n_i appears more frequently as a common hypernym of C_i, $\mathrm{freq}(n_i)$ is large. In addition, when the granularly uniform category name appears more frequently as hyponyms of n_i, $\sum \mathrm{cooccur}(n_i, c)$ is large. Therefore, $\mathrm{Granularity}(F_i)$ represents the uniformity in granularity of categories.

In this paper, entropy [13] is used as the measure of the uniformity in size of categories. Assume that each object in O composed of m objects is classified into each category f^i_j of a facet F_i. That is, f^i_j includes m_j objects. In this case, entropy $H(F_i)$ is calculated by the following formula:

$$H(F_i) = -\sum_{j=1}^{k} \frac{m_j}{m} \log \frac{m_j}{m} \quad (\sum_{j=1}^{k} m_j = m)$$

It is known that $H(F_i)$ is maximized when the same number of objects is associated with each category. Thus, $H(F_i)$ is considered to represent uniformity in the size of categories.

$\mathrm{Score}(F_i)$, the score of F_i, is determined by the following formula:

$$\mathrm{Score}(F_i) = \Phi(\mathrm{Granularity}(F_i)) \cdot \Psi(\mathrm{Rel}(n_i, q)) \cdot H(F_i)$$

$\mathrm{Score}(F_i)$ is represented by the product of $\mathrm{Granularity}(F_i)$, $\mathrm{Rel}(n_i, q)$, and $H(F_i)$. $H(F_i)$ can be at most dozens, however, the values of $\mathrm{Granularity}(F_i)$ and $\mathrm{Rel}(n_i, q)$ range from thousands to tens of thousands in some cases. Therefore,

Table 1. Queries for each category

categories	queries
regions	Barcelona, New York, Roma, Shanghai, Berlin, Hawaii, Sydney, Buenos Aires, Moscow, Cape Town
events	Halloween, Easter, Encierro, Olympic, World Cup, Nobel Prize, Academy Award, Mother's Day, Independence Day, Christmas
people	Michael Jackson, John Lennon, Picasso, George Lucas, Gandhi, Leonardo Dicaprio, Napoleon, Barack Obama, Bill Gates, Ichiro
phenomena	hurricane, tsunami, eruption, aurora, shooting star, lightning, snow, rainbow, mirage, earthquake
features	the Eiffel Tower, Sagrada Familia, Angkor Wat, the Danube, Ayers Rock, the Arc de Triomphe, Alps, the Sahara, the Great Wall, Statue of Liberty
foods	Spanish foods, Chinese foods, Turkish foods, French foods, Japanese foods, Italian foods, Indian foods, Mexican foods, Korean foods, Thai foods
company	Microsoft, Google, Disney, Toyota, Louis Vuitton, Adidas, Nintendo, Ikea, Telefonica, General Electric
animals	rabbit, dog, zebra, tiger, bear, panda, chick, dolphin, bee, kangaroo
transportation	car, yacht, plane, taxi, subway, balloon, train, sledge, hovercraft, helicopter
sports	baseball, soccer, ski, tennis, synchronized swimming, figure skating, badminton, dance, triathlon, basketball

we introduce the scaling functions $\Phi(\mathrm{Granularity}(F_i))$ and $\Psi(\mathrm{Rel}(n_i, q))$. Determining the scaling functions Φ and Ψ properly is necessary to ensure a high score for a more "*effective*" facet. A facet set $\mathcal{F}$ is composed of the top l "*effective*" facets with a high score.

6 Experiments

We conducted two experiments about generated facets and our implemented system. We describe the setting for facet generation. In these experiments, Flickr was the targeted keyword-based search system. Many tags were sufficiently associated with searched images in Flickr. The top 100 images from the search results were collected in the relevant order and used to generate the object set O. Tags of nouns and noun phrases were extracted from all tags included in the images by SStagger [15]. T comprised the top 50 most frequent tags. Sentences for the generation and validation of facets were obtained from the top 50 titles and snippets returned by Yahoo! Search BOSS API.

6.1 Effectiveness of Generated Facets

We conducted experiments to validate the effectiveness of generated facets. For a facet set $\mathcal{F}$, the relevance of a facet to an object set and the uniformity in granularity of categories were evaluated manually. With regard to the relevance of a facet to an object set, facets that are regarded as adequate for classification are called correct facets. For uniformity in the granularity of categories, we counted the number of uniform categories in the correct facets. When multiple groups of uniform categories exist, the largest number of categories in the groups is counted. In this experiment, we changed the scaling functions Φ and Ψ into linear and natural logarithmic functions, respectively. As a result, four sequences of facets were evaluated. We selected 10 categories where queries seemed probable for web image search and further selected 10 queries in each category. Table 1 shows all queries for each category. $\mathcal{F}$ is evaluated when l, the number of facets in $\mathcal{F}$, is three and five.

Table 2 shows the percentage of correct facets, and Table 3 shows the percentage of uniform categories included in correct facets. In Tables 2 and 3, "Linear" denotes a linear function and "ln" denotes a logarithmic function. In all combinations of scaling functions Φ and Ψ, the average precision of correct facets was greater than 71.7% for $l = 3$ and 62.2% for $l = 5$, as shown in Table 2.

Table 2. Percentage of correct facets included in l facets

	@3				@5			
Φ	ln	ln	Linear	Linear	ln	ln	Linear	Linear
Ψ	ln	Linear	ln	Linear	ln	Linear	ln	Linear
regions	70.0%	36.7%	53.3%	36.7%	66.0%	42.0%	58.0%	44.0%
events	76.7%	73.3%	80.0%	73.3%	74.0%	60.0%	72.0%	66.0%
people	93.3%	90.0%	90.0%	96.6%	84.0%	64.0%	84.0%	74.0%
phenomena	86.7%	80.0%	93.3%	90.0%	86.0%	70.0%	84.0%	80.0%
animals	86.7%	76.7%	83.3%	86.7%	78.0%	62.0%	72.0%	70.0%
foods	90.0%	93.3%	93.3%	96.6%	92.0%	88.0%	96.0%	94.0%
companies	83.3%	73.3%	76.7%	76.7%	76.0%	72.0%	74.0%	68.0%
transportation	83.3%	63.3%	86.7%	80.0%	72.0%	52.0%	72.0%	66.0%
features	76.7%	66.7%	90.0%	66.7%	66.0%	52.0%	74.0%	52.0%
sports	73.3%	63.3%	66.7%	63.3%	62.0%	60.0%	62.0%	62.0%
average	**82.0%**	**71.7%**	**81.3%**	76.7%	**75.6%**	**62.2%**	**74.8%**	67.6%

Table 3. Percentage of uniform categories included in correct facets

	@3				@5			
Φ	ln	ln	Linear	Linear	ln	ln	Linear	Linear
Ψ	ln	Linear	ln	Linear	ln	Linear	ln	Linear
regions	69.6%	65.5%	76.0%	72.5%	70.4%	69.0%	69.7%	73.5%
events	82.2%	82.3%	83.6%	83.9%	80.1%	79.8%	82.9%	83.3%
people	85.8%	79.2%	84.4%	84.2%	84.0%	80.9%	83.3%	83.1%
phenomena	81.1%	81.4%	80.1%	80.4%	78.5%	78.0%	79.2%	76.5%
animals	79.3%	80.4%	83.0%	77.6%	77.1%	76.9%	78.8%	76.1%
foods	87.0%	86.3%	86.2%	85.2%	84.7%	83.0%	85.1%	85.4%
companies	75.2%	73.9%	75.8%	75.6%	74.6%	74.1%	77.1%	74.0%
transportation	85.3%	87.9%	87.5%	89.3%	82.3%	83.8%	84.4%	84.4%
features	85.6%	85.4%	86.5%	86.2%	81.6%	86.1%	84.6%	86.3%
sports	77.0%	70.5%	80.9%	77.3%	75.3%	74.9%	80.2%	75.0%
average	81.0%	80.0%	82.6%	81.6%	79.0%	78.7%	80.8%	79.7%

Table 4. Five facets generated by the query "*John Lennon*"

facet name	cities	countries	songs	stars	instruments
category name	Prague New York New York City York Berlin	Czech Republic Japan France Peru Canada	Imagine Give Give peace a chance	Mick Jagger Michael Paul McCartney	guitar piano drums

Both natural logarithmic functions resulted in best performance: 82.0% for $l = 3$ and 75.6% for $l = 5$. The relevance of a facet to an object set is represented by page counts and the values ranged from thousands to tens of thousands in some cases. The relevance value could be much larger than those of Granularity(F_i) and $H(F_i)$. The natural logarithmic function reduced the effect on its score, hence, the case is considered to return good results. As shown in Table 3, there is no significant difference in the average precision of uniform categories when Φ and Ψ change.

Table 4 shows five facets in the query "John Lennon" when natural logarithmic function is adopted for both Φ and Ψ. "John Lennon" was a famous musician; thus, facets named "songs" and "instruments" are query specific. The facet "stars" was wrongly extracted. The facet was generated from the term "Paul McCartney", and the tag "Paul McCartney" was associated with images of "John Lennon." Consequently, it is necessary to eliminate beforehand coordinate terms of the input query from candidates for category names. In addition, paraphrases such as "New York" and "New York City" appeared in the "cities" facet category names. We will consider a method to detect paraphrases.

6.2 Evaluation of Faceted System for Image Search Results

In this section, we describe the experiment conducted to evaluate the effectiveness of an image search system using the proposed method.

Fig. 1 shows a screenshot of the implemented system. When a user inputs a query for an image search, the system shows the image results in the right-hand panel and dynamic facets related to the search in the left-hand panel. The user selects a facet and the category that interests him/her, and the classified images are displayed. Fig. 1 shows the results for a search for images about "Japanese foods." The images are classified in the category "sushi" for the facet "dishes."

Table 5. Description of two tasks

Task A	Select five images where “lightning is seen through clouds”.
	You are a foreigner and like Japanese foods, but you don’t know their names. Select five images of “fried food in Japan”.
	Select five images of “activities with a sledge”.
Task B	You will go to Cape Town on a trip. Select five images that interest you.
	You want images of sledges used in a presentation. Select five images that interest you.
	You will participate in Halloween for the first time. Select five images that interest you.

Table 6. Average questionnaire results

Question	Task	Proposed	Baseline
easiness	Task A	**4.27**	3.23
	Task B	**3.87**	3.17
confidence	Task A	**4.2**	3.63
	Task B	**3.87**	3.4

Table 7. Time average

Task	Proposed	Baseline
Task A	**74.0 s**	104.9 s
Task B	107.6 s	107.4 s

It is sometimes difficult for users to create queries that adequately reflect their information needs in a keyword-based search system. A user may understand his/her information needs clearly but may be unfamiliar with targeted fields and, consequently, may enter an ambiguous query. This system is thought to work well in these circumstances.

In this experiment, six participants, graduate students who regularly used search engines to locate images, searched for images in the implemented faceted system (**proposed system**) and a system that does not present facets (**baseline system**) to validate the following criteria:

1. easiness: how easily can users find images?
2. confidence: how confidently can users select images?
3. quickness: how quickly can users find images?

The participants selected five images from 100 images presented in each task. They ranked easiness and confidence on a five-point Likert scale after each task. The time required for each task was calculated to evaluate quickness. The participants could see tags with images when browsing images in both systems.

This experiment included two tasks. In Task A, a user has a clear understanding of the images he/she desires. For example, in the task for finding images where “lightning is seen through clouds,” we assume that a user cannot create an adequate query and search for the desired image (i.e., “seen through clouds”) from images of lightning returned from a search engine by an ambiguous query. In Task B, the images desired are not clearly understood. For example, given the task of finding images that would be helpful for a future trip to Cape Town, we assume that a user does not clearly understand the images to be searched. Each task is based on a query selected from 10 categories presented in Table 1. Table 5 shows shows three queries each for Tasks A and B.

Each participant performed the two tasks based on queries selected from the 10 categories for both the proposed system and the baseline system. Briefly, the participants performed five tasks by using four patterns. To consider the effect of the order of tasks, the participants were divided into two groups.

Table 6 shows the average values per task for easiness and confidence. The values for the proposed system were higher than those for the baseline system

Table 8. Queries used in faceted system for other search results

"Kyoto $\wedge$ sushi",	"Christmas $\wedge$ Seattle",	"Adidas $\wedge$ shoes",	"Messi $\wedge$ dribbling",	"Nintendo $\wedge$ action"
"Italy $\wedge$ pasta",	"Halloween $\wedge$ UK",	"Nike $\wedge$ basketball",	"Nokia $\wedge$ phone",	"Thorpe $\wedge$ freestyle"

irrespective of the type of tasks and questions. The result of the proposed system for easiness of Task A was more than that for the baseline system by one point. Hence, the proposed system is considered to work effectively.

Table 7 shows the average time required for a task. For Task A, the average time in the proposed system was approximately 30 s less than that in the baseline system. For Task B, however, a significant difference between the two systems was not evident. When users do not clearly understand the images to search, they tend to initially check all images. Thus, the effectiveness of the faceted interface was expected to decrease.

7 Faceted Navigation for Other Search Queries

In this section, we describe the use of the proposed system to navigate other related keyword queries by facets. For example, a user may use the query "Kyoto $\wedge$ sushi" to find out about sushi shops in Kyoto. If the user is satisfied with the results of this search and shifts interest to sushi shops in other cities, then he/she may not create an adequate second query. In our system, the user can move to the search results for "Tokyo $\wedge$ sushi" and "Osaka $\wedge$ sushi" by selecting the facet "cities." In this manner, the system enables navigation by displaying facets indicating the location of a search result.

To generate facets for other search queries, the term set T is composed of terms included in the keyword query. For example, facets is generated from "Kyoto" and "sushi".

We also conducted an experiment to validate the effectiveness of generated facets for other search queries. In this experiment, the relevance of a facet to an object set and the uniformity in granularity of categories were evaluated manually. The relevance of a facet to an object set is evaluated by considering correct facets, which are defined as facets desirable for navigation. Uniformity in granularity of categories is determined by an experiment similar to that described in Section 6.1. We adopted the natural logarithmic function for both Φ and Ψ because best performance was obtained by this function in the experiment previously described. Ten queries composed of two terms were used, as shown in Table 8. Only the top three facets with the highest scores were evaluated because not many facets were generated.

The percentage of correct facets was 73.3% and that of uniform categories included in correct facets was 82.0%. Similar to the facets for image search results, the generated facets are considered to be effective.

There are many categories for queries including proper names. For example, in "Halloween UK," the facet "countries" is generated and the number of categories is 17. This infers that many proper names follow the pattern "such as."

8 "Multiple-Words"-Based Facets

In this section, we discuss "multiple-words"-based facets as potential improvements of our system. Let us consider a situation wherein a user queries "Picasso Guernica" on a web search engine. Up to now, we have considered generating facets and associated categories that consist of a single word, such as facets for both "Picasso" and "Guernica." The facet for "Picasso" contains other painters' names, and the facet for "Guernica" contains other works by Picasso. Multiple words can have coordinate and super concepts. When the tuple (Picasso, Guernica) is considered, the coordinate concepts of the tuple include (Gogh, Sunflowers) and (Goya, Colossus), where each tuple is a pair consisting of a great painter and his outstanding works.

Obtaining related terms for a set of words is hugely challenging, and as yet, no methodology has been developed. Hence, we propose a method to obtain coordinate concepts for a given set of words on the fly. Because of the combinatorial explosion of words, it is impossible to construct dictionaries for sets of words in advance.

Here, the purpose is to obtain coordinate terms for a multiple-word set. For example, when a word set {Kyoto, Geisha} is given as the input to the proposed algorithm, it can return {Tokyo, Sushi}, {Nara, Great Buddha}, {Mt. Fuji}, and {Japanese sword} because they are all symbols of traditional Japanese culture. The input to the algorithm is a word set consisting of n words, and the output is a list of word sets. In this paper, we only focus on finding {Tokyo, Sushi} and {Nara, Great Buddha} in this example, where any of the target word sets can be regarded as a tuple consisting of a location name and a cultural element. We have not considered the problem of finding other types of coordinate terms, such as {Mt. Fuji} and {Japanese sword}.

For ease of explanation, we will describe a case where two words are input, although our proposed algorithm does not limit the number of words. The input to the algorithm is (x_0, y_0), and the algorithm consists of the following steps:

1. Collect text resources using a web search engine.
   In the case of two input words, four queries are issued to a web search engine. Assume that the input is (Picasso, Guernica); then the queries are "or Picasso $\wedge$ Guernica," "Picasso or $\wedge$ Guernica," "Picasso $\wedge$ or Guernica," and "Picasso $\wedge$ Guernica or." Titles and snippets of 100 search results are collected for each query. Our proposed method does not use any other resource. That is, the number of web accesses is limited to enable the procedure to be performed quickly.
2. Obtain the results of $\textbf{sibling}_{\Leftarrow}(x_0)$ and $\textbf{sibling}_{\Rightarrow}(x_0)$. A term set $X := \{x | x \in \textbf{sibling}_{\Leftarrow}(x_0) \wedge x \in \textbf{sibling}_{\Rightarrow}(x_0)\}$ is obtained, where X is a set of coordinate terms of x_0.

   The function $\textbf{sibling}_{\Leftarrow}(z)$ returns all text strings that appear just **before** "*or z*" in the collected resources, and the function $\textbf{sibling}_{\Rightarrow}(z)$ returns all text strings that appear just **after** "*z or.*" In the above example, if both "Gogh or Picasso" and "Picasso or Gogh" appear, "Gogh" is considered to be an element of X and it is regarded as a coordinate term of "Picasso."

3. Obtain the results of $\mathbf{sibling}_{\Leftarrow}(y_0)$ and $\mathbf{sibling}_{\Rightarrow}(y_0)$. A term set $Y := \{y|y \in \mathbf{sibling}_{\Leftarrow}(y_0) \wedge y \in \mathbf{sibling}_{\Rightarrow}(y_0)\}$ is obtained, where Y is a set of coordinate terms of y_0.
4. Obtain the result of $\mathbf{connector}(x_0, y_0)$, and $\mathbf{connector}(y_0, x_0)$. They are syntactic patterns frequently (more than twice) appearing between $x \in X$ and $y \in Y$. A member of $\mathbf{connector}(x_0, y_0)$ is denoted by $c_{x \to y}$, and a member of $\mathbf{connector}(y_0, x_0)$ is denoted by $c_{y \to x}$.

   In the above example, if "Picasso painted Guernica" appears more than twice, $c_{x \to y}$ can be "painted." If "Guernica by Picasso" appears more than twice, $c_{y \to x}$ can be "by."
5. Obtain the results of $\mathbf{related}_{\Leftarrow}(c_{x \to y}, y_0)$ and $\mathbf{related}_{\Rightarrow}(c_{y \to x}, y_0)$. A term set $\{x|x \in (\mathbf{sibling}_{\Leftarrow}(x_0) \cup \mathbf{related}_{\Leftarrow}(c_{x \to y}, y_0)) \wedge x \in (\mathbf{sibling}_{\Rightarrow}(x_0) \cup \mathbf{related}_{\Rightarrow}(c_{y \to x}, y_0))\}$ is added to X.
   In this step, the method tries to find coordinate terms of x_0 by using y_0 and connectors.
6. Obtain the results of $\mathbf{related}_{\Leftarrow}(c_{y \to x}, x_0)$ and $\mathbf{related}_{\Rightarrow}(c_{x \to y}, x_0)$. A term set $\{y|y \in (\mathbf{sibling}_{\Leftarrow}(y_0) \cup \mathbf{related}_{\Leftarrow}(c_{y \to x}, x_0)) \wedge y \in (\mathbf{sibling}_{\Rightarrow}(y_0) \cup \mathbf{related}_{\Rightarrow}(c_{x \to y}, x_0))\}$ is added to Y.

   In this step, the method tries to find coordinate terms of y_0 by using x_0 and connectors. For example, if both "Picasso painted Bullfight" and "Bullfight by Picasso" appear, "Bullfight" is added to Y. When "Picasso painted Bullfight" appears but "Bullfight by Picasso" does not appear, if "Bullfight or Guernica" appears, "Bullfight" is still added to Y.
7. By using any combination of $x' \in X$ and $y' \in Y$ instead of (x_0, y_0), the method executes steps 2, 3, 5, and 6.
8. First, the method generates any combination of $x' \in X$ and $y' \in Y$. For a pair (x', y'), text strings using $c_{x \to y}$ and $c_{y \to x}$ are constructed. That is, "$x' c_{x \to y} y$" and "$y' c_{y \to x} x$" are constructed. The number of appearances of the text strings is counted as the score of the pair (x', y'), and a pair whose score is more than zero is contained in the output.

   For example, if "Goya" is in X, "Colossus" is in Y, "Goya painted Colossus" appears once, and "Colossus by Goya" appears twice, the pair (Goya, Colossus) is contained in the output and its score is three.

When the query is (Picasso, Guernica), the actual output contains (Dali, Premonition of Civil War), (Gogh, Sunflowers), (Goya, Colossus), (Goya, The Nude Maja), and (Monet, Nympheas). However, the results contain some "noise" and the method still needs to be improved.

9 Conclusions

We have proposed a method for generating facets on-the-fly to enhance navigation for objects returned by keyword searches. Facets are generated by constructing pairs of a hypernym and coordinate terms of candidates of category names. The generated facets are ranked so that they can be effectively used for browsing search results. We conducted an experiment to assess the facets. We also implemented a system that shows the result of an image search classified by multiple facets, and a user study showed that the performance of the system is good. Furthermore, we have proposed a method of generating "multiple-words"-based facets. We will study the generation of a set of orthogonal facets in future work.

Acknowledgements. This work was supported in part by the Kyoto University Global COE Program entitled "Informatics Education and Research for Knowledge-Circulating Society."

References

1. Basu Roy, S., Wang, H., Das, G., Nambiar, U., Mohania, M.: Minimum-effort driven dynamic faceted search in structured databases. In: Proc. CIKM 2008, pp. 13–22 (2008)
2. Cui, H., Wen, J.R., Nie, J.Y., Ma, W.Y.: Probabilistic query expansion using query logs. In: Proc. WWW 2002, pp. 325–332 (2002)
3. Dash, D., Rao, J., Megiddo, N., Ailamaki, A., Lohman, G.: Dynamic faceted search for discovery-driven analysis. In: Proc. CIKM 2008, pp. 3–12 (2008)
4. Furnas, G., Landauer, T., Gomez, L., Dumais, S.: The vocabulary problem in human-system communication. Communications of the ACM 30, 964–971 (1987)
5. Ghahramani, Z., Heller, K.: Bayesian sets. In: Proc. NIPS 2005, pp. 435–442 (2005)
6. Hearst, M.: Search User Interface. Cambridge University Press (2009)
7. Hearst, M.: Automatic acquisition of hyponyms from large text corpora. In: Proc. Coling 1992, pp. 539–545 (1992)
8. Li, C., Yan, N., Roy, S., Lisham, L., Das, G.: Facetedpedia: dynamic generation of query-dependent faceted interfaces for wikipedia. In: Proc. WWW 2010, pp. 651–660 (2010)
9. Lim, S., Liu, Y., Lee, W.: Faceted search and retrieval based on semantically annotated product family ontology. In: Proc. WSDM 2009 Workshop on Exploiting Semantic Annotations in Information Retrieval, pp. 15–24 (2009)
10. Miller, G.: WordNet: a lexical database for English. Communications of the ACM 38(11), 39–41 (1995)
11. Ohshima, H., Tanaka, K.: Real time extraction of related terms by bi-directional lexico-syntactic patterns from the web. In: Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication, pp. 441–449 (2009)
12. Pound, J., Paparizos, S., Tsaparas, P.: Facet discovery for structured web search: a query-log mining approach. In: Proc. SIGMOD 2011, pp. 169–180 (2011)
13. Shannon, C., Weaver, W.: The mathematical theory of communication (1959)
14. Shinzato, K., Torisawa, K.: A simple www-based method for semantic word class acquisition. In: Proc. RANLP 2005, vol. 292, p. 207 (2007)
15. Tsuruoka, Y., Tsujii, J.: Bidirectional inference with the easiest-first strategy for tagging sequence data. In: Proc. of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing, pp. 467–474 (2005)
16. Tunkelang, D.: Dynamic category sets: An approach for faceted search. In: SIGIR Workshop on Faceted Search (2006)
17. Wen, J.R., Nie, J.Y., Zhang, H.J.: Clustering user queries of a search engine. In: Proc. WWW 2001, pp. 162–168 (2001)
18. Yee, K., Swearingen, K., Li, K., Hearst, M.: Faceted metadata for image search and browsing. In: Proc. SIGCHI 2003, pp. 401–408 (2003)

Searching for Quality Microblog Posts: Filtering and Ranking Based on Content Analysis and Implicit Links

Jan Vosecky, Kenneth Wai-Ting Leung, and Wilfred Ng

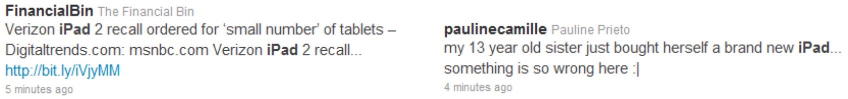
Department of Computer Science and Engineering
Hong Kong University of Science and Technology, Hong Kong, China
{jvosecky,kwtleung,wilfred}@cse.ust.hk

Abstract. Today, social networking has become a popular web activity, with a large amount of information created by millions of people every day. However, the study on effective searching of such social information is still in its infancy. In this paper, we focus on Twitter, a rapidly growing microblogging platform, which provides a large amount, diversity and varying quality of content. In order to provide higher quality content (e.g. posts mentioning news, events, useful facts or well-formed opinions) when a user searches for tweets on Twitter, we propose a new method to filter and rank tweets according to their quality. In order to model the quality of tweets, we devise a new set of link-based features, in addition to content-based features. We examine the implicit links between tweets, URLs, hashtags and users, and then propose novel metrics to reflect the popularity as well as quality-based reputation of websites, hashtags and users. We then evaluate both the content-based and link-based features in terms of classification effectiveness and identify an optimal feature subset that achieves the best classification accuracy. A detailed evaluation of our filtering and ranking models shows that the optimal feature subset outperforms traditional bag-of-words representation, while requiring significantly less computational time and storage. Moreover, we demonstrate that the proposed metrics based on implicit links are effective for determining tweets' quality.

1 Introduction

In recent years, social networking and microblogging services have seen a steep rise in popularity, with users from a wide range of backgrounds contributing content in the form of short text-based messages. Microblogging services, in particular Twitter, are at the epicentre of the social media explosion, with millions of users being able to create and publish short messages, referred to as *tweets*, in real time. It is estimated that nearly 200 million tweets are generated and over 1.6 billion search queries are issued each day [1] and these figures are likely to keep rising in future. However, the work on searching tweets or similar social information is still in its infancy. Unlike traditional web search, the search results from social networking services may be mostly relevant to the query, however may also include a large proportion of low-quality and noisy messages.

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 397–413, 2012.
© Springer-Verlag Berlin Heidelberg 2012



FinancialBin The Financial Bin
Verizon **iPad** 2 recall ordered for 'small number' of tablets – Digitaltrends.com: msnbc.com Verizon **iPad** 2 recall... http://bit.ly/iVjyMM
5 minutes ago

paulinecamille Pauline Prieto
my 13 year old sister just bought herself a brand new **iPad**... something is so wrong here :|
4 minutes ago

Fig. 1. Two relevant tweets returned as a result to a search for 'iPad'. The first tweet shares factual news about the product, while the second tweet only mentions about the author's family and includes an unclear subjective judgement.

In this paper, we focus on Twitter, a popular social networking and microblogging platform. The social networking features include subscribing to tweets by other users, forwarding tweets from other users and explicitly addressing other users in their tweets. During recent events, such as natural disasters or political turbulences, the influence of Twitter has become even more evident.

While there has been plenty of study on the dynamics of information spreading, influence and authority in Twitter, little attention was paid to how to find good quality content in Twitter. Twitter is clearly a rich source of data, however, there are also several new challenges compared with traditional web searching.

- Very brief content: In Twitter, only 140 characters are available to convey the author's message. This poses new challenges to establish an effective set of features for filtering and ranking search results.
- Highly dynamic in scale: There is a large quantity of such postings, with nearly 200 million new tweets published each day [1]. This demands more efficient techniques for identifying high quality tweets.
- Informal language: Being user-generated content, postings often contain misspellings, abbreviations, slang expressions and the like. This makes the analysis of tweets more difficult.
- Varying quality: The level to which a tweet contains high quality information varies dramatically. [2] found that 57% of tweets are not of general interest, except to the author or the author's close friends. [3] claims that users post several types of messages, only some of which are intended to be of interest to a wider audience.

The above challenges become more obvious when searching for content in Twitter, which presents results ordered by recency of posting. The user then needs to manually pick out high quality content among potentially thousands of results. Consider a search for the term "iPad" and two different search results shown in Figure 1. The example illustrates that there may be a large difference in the level of quality of tweets returned as results to a search query.

In order to achieve more effective tweet search, we tackle the following two problems: filtering[1] and ranking tweets according to their quality. We may apply these two methods on a recent set of tweets. Basically, our approach involves

[1] The terms "filtering" and "classification" are used interchangeably throughout the paper.

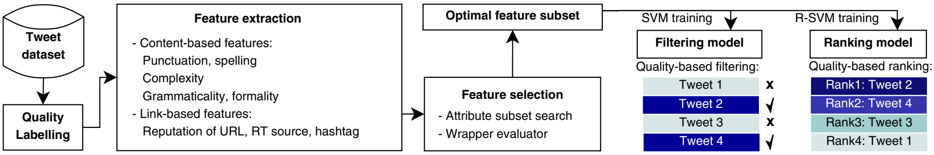


Fig. 2. Overall process of filtering and ranking tweet based on quality analysis

the following steps, as illustrated in Figure 2. First, to capture the quality of individual tweets, we examine each tweet with a set of content-based and link-based features. Our newly proposed link-based features leverage the implicit relationships between tweets, hashtags[2] and hyperlinks. Second, we evaluate the effectiveness of individual features and perform attribute subset selection to obtain an optimal subset of features for the classification task. Third, the optimal feature subset can be used for constructing (i) a filtering model, or (ii) a ranking model. The filtering model aims to identify high quality tweets from a given set of tweets, or to filter out low quality, noisy tweets. On the other hand, the ranking model aims to rank a set of tweets based on their quality, with high quality tweets ranked at top positions. Finally, we show that our optimal feature subset outperforms a baseline method based on TFIDF representation for the filtering task, while requiring less computational time and storage in our empirical evaluation. In addition, for the ranking task, we significantly outperform a state-of-the-art method based on hyperlink presence.

Our contributions in this paper are then as follows:

- We propose a new strategy for filtering and ranking of tweets, focusing on the quality of a tweet, in order to improve on the basic search functionality in Twitter. Our detailed evaluation shows that our strategy helps to improve the naïve recency-as-relevance approach currently used.
- We propose a novel set of link-based features in order to model the quality of a tweet, utilizing the implicit relationships between tweets, hyperlinks and users. We introduce three metrics to reflect the quality of tweets which relate to a specific URL, hashtags or a user. These features provide useful evidence to our models and boost the filtering and ranking performance.
- We examine both link-based and conventional content-based features, and evaluate their effectiveness in the modelling task. We then identify an optimal (best-performing) feature subset. To our knowledge, this is the first study with detailed analysis of features for the task of filtering and ranking of general tweets according to their quality.

The remainder of the paper is organized as follows. Section 2 discusses related work in the area of Twitter analysis. In Section 3, we present background on Twitter and define the notion of tweet quality. Section 4 discusses our filtering

[2] Hashtags are tags prefixed with a '#' symbol to indicate the topics of the tweet and enable posts related to the same topics to be quickly searched.

approach and the features of tweets. Our ranking approach is then presented in Section 5. Section 6 provides an evaluation of the filtering and ranking models, as well as of our proposed features. Finally, Section 7 concludes the paper.

2 Related Work

Twitter has been an active area of research in recent years. [3,4] provide initial insights into the usage patterns in Twitter and how communities are formed. From the content point of view, previous work largely aimed to classify tweets into a predefined set of categories, based on their purpose (such as 'news', 'events', 'opinions', etc.). [5] proposes a feature-based method for classifying tweets into 6 categories. [6] proposes the use of topic models and supervised learning to assign 4 broad topics to tweets. Our work is complementary to these works as we focus on the quality of tweets, which is a new dimension orthogonal to such predefined categories.

From the perspective of analyzing the quality of tweets, [2] is closer to our work. It provides initial insights in the classification of Tweets based on their interestingness to the reader and presents a set of potential features. However, only the presence of a hyperlink is used for classification in [2]. Our work focuses on a more generalized quality-based classification and ranking, examine a larger set of features and proposes new features which outperform the link-only approach. We also provide a detailed feature evaluation. In [7], several features are proposed to find interesting clusters of tweets for specific events but they do not analyse the post's content. Instead, cluster size, expected audience and time span of the cluster are the criteria for gauging quality.

From the perspective of effective tweet ranking, early attempts at new algorithms to rank tweets were proposed. [8] proposes several simple methods, e.g. based on the number of followers of a user or the length of a tweet. [9] ranks tweets using non-negative matrix factorization, based on the bag-of-words representation. However, these works did not provide comprehensive evaluation or convincing empirical results. [10] ranks tweets in Twitter-like forums based on star-ratings or thumb-ratings, not taking content into account. [11] employs a learning-to-rank approach using a hybrid set of features (query-content relevance, content-based features, author features). In our work, we focus on the specific problem of analyzing the *quality* of an individual tweet. We formulate criteria of tweet quality and examine a comprehensive set of content-based and novel link-based features. Our work also provides a detailed feature evaluation for tweet filtering and ranking based on their quality.

3 Tweet Quality Analysis

3.1 Preliminaries

Twitter is a social networking and microblogging platform, in which registered users may post short messages (*tweets*) of up to 140 characters in length. These

Table 1. Examples of tweets judged by different quality criteria

Criterion	*Positive example*	*Negative example*
Well-formedness	"Lady Gaga is on the 4th place among solo artists with the most top tens in a row, only behind Janet Jackson, Madonna and Whitney Houston."	"Serena got a $2000 fine for the outburst....hahhahahaahahhaha but she told her her news yong! LOL"
Factuality	"Apple to release iOS 5 GM to assemblers during week of Sept. 23 (@thisisneil / AppleInsider)"	"so now that i have my iphone is jailbroken, what should i download on it ? i really dont know"
Navigational quality	"#Japan's prime minister promises help to city decimated by tsunami and earthquake http://dlvr.it/N2v7q" *[links to a news article]*	"This is what I call a perfect Sunday afternoon! http://bit.ly/endbUc" *[links to a family photograph]*

messages are published and available for search in near-real time and can be posted either using a web interface, via SMS messages or through a wide range of third-party applications. Currently, Twitter has over 300 million registered users who post over 200 million tweets and submit around 1.6 billion search queries per day [12].

The social networking features include: (1) subscribing to posts by another user (*follow*), (2) forwarding posts from other users (*re-tweet*, indicated by a "RT" prefix) and (3) explicitly addressing users in their posts (*mentions*, indicated by a "@" symbol followed by a username), thus enabling conversations and replies to be carried out. Within tweets, users may also include *hashtags* (tags prefixed with a '#' symbol) to indicate the topics discussed and enable posts related to the same topics to be grouped together and searched more directly.

By default, the user's profile and tweets are publicly accessible, unless restricted to the user's *followers*. Data available on Twitter is also accessible via Twitter's REST API.

3.2 Goal Definition: Defining Tweet Quality

In this section, we focus on the notion of *tweet quality* more closely and set out our goals for modelling and assessing the quality of tweets. Based on their purpose, messages on Twitter have been found to fall into several categories, such as conversational, information sharing, news reporting, etc. [3]. Instead of focusing on a specific type or category of tweets, we aim to establish criteria for judging the quality of tweets in general. Therefore, we define our notion of an 'interesting' tweet along the following 3 criteria:

- *Well-formedness.* Well-written, grammatically correct and understandable tweets are preferred over tweets containing heavy slang, uncomprehensible language or excessive punctuation.
- *Factuality.* News, events, announcements and other facts of general interest are preferred over tweets with an unclear message, private conversations and generic personal feelings, which typically do not convey useful information.

– *Navigational quality.* A tweet that links to reputable external resources (e.g. news articles, reports, or other online materials) may provide further information to the reader. However, not all links may be of general interest (e.g. links to photo sharing websites, used for sharing personal photos). Therefore, it is important to distinguish what type of website a tweet refers to.

Examples of tweets judged along these criteria are shown in Table 1. In real scenarios, tweets may exhibit more than one of the criteria (e.g. news-oriented tweets are typically factual and provide a link to the full news article). In fact, these 3 criteria allow for flexibility when judging different types of tweets[3].

In order to assess tweets according to the quality criteria, we follow a process described in subsequent sections. In particular, we extract features from tweets (Section 4.2) to capture various characteristics, as inspired by the 3 criteria described in this section. These features then form a basis of our filtering and ranking models.

4 Quality-Based Tweet Filtering

4.1 Classification Method

Since our work focuses on the tweet-specific feature extraction and evaluation, rather than on the classification algorithm itself, we utilize standard classification tools. Due to its wide-spread adoption and proven effectiveness in text mining tasks, we use *Support Vector Machines* (SVM) for the classification task.

4.2 Characterizing Tweets with Features

To gain deeper insight into which factors most influence the quality of a tweet, we extract a number of features from every tweet. The features can be broadly divided into *content-based* and *link-based* features. *Content-based* features may be used to identify low-quality tweets which contain many spelling mistakes and use punctuation excessively. These features correspond to the *well-formedness* criteria in Section 3.2. Next, features based on the complexity and formality of the language correspond to the *factuality* criteria. *Link-based* features include the presence of hyperlinks, hashtags or mentions of other users. We also propose a set of novel metrics to obtain reputation scores for URL domains, users and hashtags. These features, in particular the URL domain reputation, addresses the *navigational quality* criteria.

Punctuation and Spelling Features

Excessive Punctuation. We measure any abnormalities in punctuation with features, such as the number of exclamation marks, number of question marks and the maximum number of repeated characters.

[3] For example, when searching for tweets reviewing a movie, ‘well-formed’ tweets would be preferred over those containing excessive slang or strong language. Or, when searching for tweets about ‘iPhone’, tweets linking to news articles about ‘iPhone’ would be preferred over tweets linking to private photos taken with an ‘iPhone’.

Capitalization. Another kind of abnormality is content written in all-capitalized letters. We capture the presence of all-capitalized words and the largest number of consecutive words in capital letters.

Spelling. We extract the number of correctly spelled words and the percentage of words found in a dictionary. The dictionary used in this task is provided by the Stanford Natural Language Processing lab.

Syntactic and Semantic Complexity

Syntactic Complexity. We measure the absolute length of the tweet, average word length, maximum word length and the percentage of stopwords. We also determine whether specific symbols, such as emoticons are present. The presence of numbers and measure symbols ($, %) is also extracted, and would apply to tweets that mention specific monetary or statistical data.

Tweet Uniqueness. On a higher level, we measure the uniqueness of a tweet relative to other tweets by the same author. This feature is based on the traditional TFIDF approach in information retrieval. We may view a tweet t_j as a set of terms $t_j = \{w_1, \ldots, w_n\}$. The uniqueness of a tweet t_j is then defined as $uniq(t_j) = \sum_{w_i \in t_j} tf_{i,t_j} \times idf_i$, where tf_{i,t_j} is the frequency of term i in tweet j and idf_i is the inverse document frequency of term i. More specifically, $tf_{i,t_j} = \frac{n_{i,t_j}}{\sum_k n_{k,t_j}}$ where n_{i,t_j} is the number of occurrences of term i in tweet j. The inverse document frequency of term i is defined as $idf_i = \log \frac{|T_u|}{|\{t_k : w_i \in t_k\}|+1}$ where $|T_u|$ is the total number of tweets from user u and the denominator indicates the number of tweets containing term i.

Grammaticality

Parts-of-Speech. We analyze parts-of-speech (PoS) within the tweet (such as nouns, verbs, adjectives, etc.). We use a PoS tagger [13] to tag each word within the tweet with its corresponding PoS. We also check whether first-person parts-of-speech are present.

From existing readability metrics to measure the complexity and formality of written text, we chose the "formality score" from [14], which is based on the amount of different PoS that occurs in a text. The score is typically used to estimate the difficulty of understanding longer pieces of text, such as articles or books. The formality score[4] is defined as:

$$\begin{aligned}F = ((&nounfrequency + adjectivefreq. + prepositionfreq. + articlefreq. - \\ &pronounfreq. - verbfreq. - adverbfreq. - interjectionfreq. + \lambda)/2)\end{aligned} \tag{1}$$

Presence of Names. We identify proper names within the tweet as words with a single initial capital letter. We also determine the maximum number of consecutive proper names in the tweet.

[4] The formality score was originally designed for longer pieces of text, with $\lambda = 100$. We adapt the value of λ in order to match the restricted length of tweet messages, $\lambda = 10$.

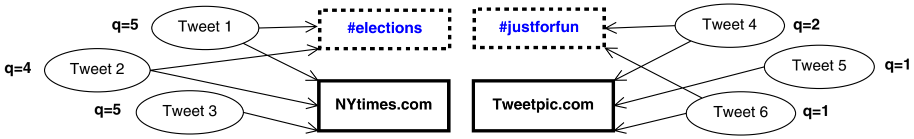


Fig. 3. Illustration of tweets containing links to two URL domains and two hashtags, including the tweets' quality scores (q)

Next, we identify named entities in the tweet using a Named Entity Recognition (NER) tagger [15]. The tagger labels words or word groups which are likely to refer to names of places, persons or organizations.

Link-Based Features

Out-link Features. We extract whether the tweet contains a hyperlink, if it is a re-tweet (indicated by the "RT" prefix), the number of '@username' mentions within tweet and the number of '#hashtags'.

Reputation Features. One observation made about tweets that contain links is that tweets which link to specific web sites, such as news portals, generally contain higher quality information than tweets which link to domains such as social networking or picture sharing web sites. We generalize this problem to any URL domain and propose a feature that captures the reputation of the domain, based on the quality of tweets that point to that domain. A URL domain should have a high reputation score if (1) many tweets link to the domain, and (2) the tweets are of good quality. Conversely, if many low-quality tweets link to a domain, its reputation score should be low.

We then extend this concept to hashtags and re-tweeted users. The re-tweet based reputation of a user captures the quality of re-tweets originally posted by that user. The intuition is that a user should have a high reputation score if his or her tweets have been (1) re-tweeted often and also (2) are of good quality. For hashtags, the reputation score is based on the quality of tweets containing a specific hashtag. Figure 3 shows an example of two groups of tweets that link to two websites, with some tweets also containing hashtags. Our focus is on what can be generalized about the websites and hashtags from the fact that Tweets 1-3 have a higher quality than Tweets 4-6.

URL Domain Reputation. As mentioned earlier, the purpose of the URL domain reputation feature is to capture the quality of tweets that link to a specific URL domain[5].

To calculate the URL reputation, we firstly define an average domain quality measure. For a set $T_d = \{t \in T : t \mapsto d\}$ consisting of tweets that link to domain d, the Average Domain Quality ($AvgDQ$) is given by:

[5] The process of extracting the feature requires two pre-processing steps. First, URL links posted in microblogs are commonly shortened to save space in the post, resulting in the need to translate shortened links to their original URL. Second, we group each tweet containing a link to the respective first-order domain of the URL.

Table 2. URL domains with the highest and lowest Domain Reputation Score (*DRS*)

10 Domains with Highest DRS				**10 Domains with Lowest DRS**			
Domain	*Inlinks*	*AvgDQ*	*DRS*	*Domain*	*Inlinks*	*AvgDQ*	*DRS*
gallup.com	99	0.96	1.92	tweetphoto.com	126	-0.86	-1.80
mashable.com	101	0.76	1.53	twitpic.com	140	-0.80	-1.72
hrw.org	58	0.86	1.52	twitlonger.com	58	-0.93	-1.64
shoppingblog.com	47	0.87	1.46	lockerz.com	54	-0.81	-1.41
redcross.org	30	0.80	1.18	yfrog.com	93	-0.70	-1.38
intuit.com	61	0.57	1.02	laurenconrad.com	33	-0.88	-1.33
good.is	31	0.68	1.01	celebuzz.com	19	-1.00	-1.28
usa.gov	30	0.67	0.98	myloc.me	24	-0.83	-1.15
thegatesnotes.com	24	0.67	0.92	instagr.am	54	-0.63	-1.09
reuters.com	8	1.00	0.90	formspring.me	20	-0.80	-1.04

$$AvgDQ(d) = \frac{1}{|T_d|} \sum_{t \in T_d} q_t, \tag{2}$$

where q_t denotes the quality score of tweet t, $q_t \in [-1, +1]$.

We then use this measure to define the Domain Reputation Score (*DRS*) as:

$$DRS(d) = AvgDQ(d) \times \log(|T_d|), \tag{3}$$

where $AvgDQ(d) \in [-1, +1]$.

Intuitively, *DRS* formalizes the idea that the reputation score is increasing with more high-quality tweets linking to d and vice versa. To illustrate the reputation scores obtained using this approach, we calculate *DRS* for all URL domains in our dataset. Table 2 lists domains with the highest and the lowest *DRS*, the *AvgDQ*, and the number of tweets linking to the domain (Inlinks)[6].

RT Source Reputation. Similarly to URL domain reputation, we leverage the quality of re-tweets that originate from a specific user in order to obtain the source user's reputation.

The RT Source Reputation Score (*RRS*) is given by: $RRS(u) = \left[\frac{1}{|RT_u|} \sum_{t \in RT_u} q_t\right] \times \log(|RT_u|)$, where RT_u is the set of re-tweets originally posted by user u and $q_t \in [-1, +1]$ is the quality score of tweet t.

Hashtag Reputation. Hashtag reputation leverages the quality of tweets related to a particular hashtag. The Hashtag Reputation Score (*HRS*) is calculated as: $HRS(h) = \left[\frac{1}{|T_h|} \sum_{t \in T_h} q_t\right] \times \log(|T_h|)$, where T_h is the set of tweets including hashtag h and $q_t \in [-1, +1]$ is the quality score of tweet t.

Timestamp. We use two features based on the timestamp of the tweet. The timestamp is discretized by hour of the day, as well as day of the week.

[6] Among the top 10 domains, there are 4 news-related sites, the website of the Red Cross and Human Rights Watch and 2 popular blogs.

5 Ranking Tweets by Quality

One of the drawbacks of the filtering method proposed in Section 4 is that it may not always be possible to clearly determine which class a particular tweet belongs to. This is true especially for tweets close to the classification boundary. Intuitively, such tweets could be labelled as being "average quality" or "neutral". While multiple classes of quality could be introduced, their exact meaning would be hard to define or interpret. A more intuitive solution might be to assign a continuous-valued score to a tweet (given by a regression model), or to produce a ranking for a set of tweets.

The goal of our ranking approach is to order a set of tweets based on their relative quality. More specifically, the ranking is based on the quality when considering each pair of tweets in the dataset. Our aim is to find a function $\mathcal{F}$ which, given two tweets t_1 and t_2, would output an ordered pair $\mathcal{F}(t_1, t_2) = (t_1 \succ t_2)$ iff $q_{t_1} > q_{t_2}$. In this way, given a set of tweets, we can produce an ordered sequence based on their quality.

5.1 Ranking Method

Our general approach proceeds in three phases: (1) tweets matching a query (based on string matching) are retrieved, (2) features of the tweets are extracted (as presented in Section 4.2) and (3) the query-tweet pairs, together with the quality scores of the tweets, are passed as input to a Learning-to-rank algorithm.

We adopt Rank SVM [16] to construct our ranking model, which is a simple and widely used Learning-to-rank technique. It takes pair-wise relationships between queries and tweets with their corresponding quality labels to learn a ranking model. Given an input set of unordered instances, the model will then output a sequence of instances ordered by their relative quality.

5.2 Features for Ranking

Similarly to our filtering approach, the criteria for ranking are based on the quality of a tweet. For this reason, we adopt the same set of features as presented in Section 4.2 to describe the content-based and implicit link-based characteristics of tweets for our ranking model.

6 Experimental Evaluation

In this Section, we describe our evaluation dataset and present the results of our filtering and ranking methods, with a particular focus on feature importance. We illustrate the overall evaluation flow in Figure 4.

6.1 Dataset

The dataset used in our experiments consists of 10,000 tweets from 100 Twitter users, with 100 recent tweets from each user. The dataset is collected from two

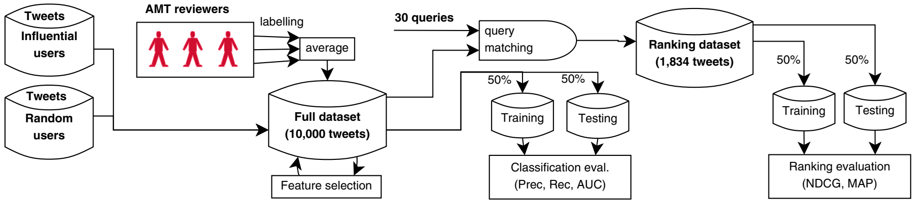


Fig. 4. Evaluation flow diagram

different user classes, namely *general users* and *influential users*. The first user class contains 50 randomly selected users from a Twitter dataset provided by [17]. These users represent members of the general public. The second user class contains 50 influential users, selected from a popular website[7] that lists influential Twitter users in various categories. The 50 users are randomly selected from 5 different categories (technology, business, politics, celebrities and activism) to avoid any topical bias and their tweets were crawled using Twitter's REST API.

Training Data Labelling. Due to a lack of a publicly available Twitter dataset with quality judgments, we manually build an evaluation dataset. To obtain the quality labels for our Twitter dataset, we utilize the Amazon Mechanical Turk[8] crowdsourcing service. The collected tweets are presented in a random order to reviewers, who are asked to assign a 1-5 rating to each tweet. Rating "1" represents a low-quality tweet, while rating "5" represents a high-quality tweet. To increase the objectivity of labelling and avoid bias from any individual reviewer, the ratings are collected and averaged from three different reviewers.

After labelling the Twitter dataset, we analyse the distribution of tweet quality in the dataset (Figure 5). Apart from the overall distribution, we also extract quality distributions for the two different user classes, general users (5,000 tweets) and influential users (5,000 tweets). Furthermore, we also analyze the distributions for re-tweets (1,941 tweets) and reply-tweets (2,676 tweets). Based on the results in Figure 5, we observe that the proportion of high-quality tweets from influential users is considerably larger than those from random users. We also observe that influential users may sometimes post low-quality tweets, thus the proposed filtering and ranking methods are also useful for tweets written by influential authors. Interestingly, we find that re-tweets are only slightly better than general tweets in terms of quality, indicating that even re-tweets may include low-quality or noisy messages. Finally, we observe that reply tweets have mostly low quality, most likely due to their conversational nature.

6.2 Filtering Evaluation

Evaluation Methodology. To evaluate our filtering method, we use 50% of randomly selected tweets from our labeled dataset as the training set and the

[7] http://www.listorious.com

[8] https://www.mturk.com

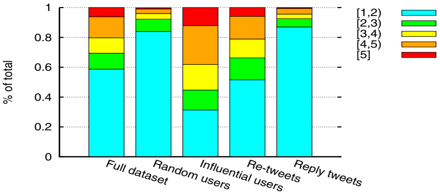


Fig. 5. Distribution of tweet quality as average quality ratings assigned by reviewers (higher rating implies higher quality)

remaining 50% as the test set. Since the labeled ratings are in the range of $[1, 5]$, they are converted to binary labels based on the mean value 3 ($label \leq 3$ meaning 'low-quality', $label > 3$ meaning 'high-quality'). The SVM classifiers are trained using the features as discussed in Section 6.2. In addition to the proposed features, we also extract n-grams up to length 5 ($1 \leq n \leq 5$) from the tweets and use a TFIDF representation as an opponent in the comparison. In the evaluation, we use standard precision and recall with respect to each of the binary labels. We also present the Area under the ROC Curve (AUC) as an overall performance metric in the comparsion.

Feature Selection. To study the importance of each feature for the classification task, we first calculate Information Gain (IG) of each feature with respect to the class label. Table 4 lists all features sorted by their IG values. We observe that the top two link-based features ($IG = 0.374$ and 0.287) significantly outperform other features ($IG \leq 0.130$) in terms of IG, showing that they are the two most important features in the classification. Moreover, language complexity and named entities are also very useful in the classification with a high IG.

The next goal is to identify the optimal subset of features for the SVM classification model. For that purpose, we employ Greedy Forward attribute subset search and a Wrapper evaluator [18] as the feature selection algorithm. Greedy Forward attribute search starts with an empty set of features and greedily adds new features which contribute most to the classification. The search finishes once the newly added feature would no longer affect the performance of the classification. To pick the optimal subset of features, a Wrapper evaluator is used. Wrappers are used to measure classification performance based on a particular subset of features. In our experiments, SVM is used as the learning scheme and the performance of each feature subset is evaluated using 2-fold cross-validation on the training dataset. The optimal 15 features are shown in Table 3.

We can see that $\frac{1}{3}$ of the optimal features are linked-based, showing that the proposed link-based features (corresponding to the *navigational quality* criteria, Section 3.2) contribute most to the classification, a conclusion also derived from the classification results (Section 6.2). Also, language formality and complexity features (corresponding to the *factuality* criteria) are strong indicators, as high quality tweets tend to use more formal language, named entities, etc. Some of

Table 3. Feature subset selected by greedy attribute subset search and SVM-wrapper

Domain reputation	RT source reputation	No. named entities
Formality	Tweet uniqueness	% correct. spelled words
Max. no. repeat. letters	Contains numbers	No. capitalized words
No. hash-tags	No. exclam. marks	Avg. word length
Contains first-person	Is re-tweet	Is reply-tweet

Table 4. Importance of individual features based on Information Gain (IG)

IG	*Feature*	*IG*	*Feature*	*IG*	*Feature*
0.374	Domain reputation	0.078	Day in the week	0.014	Contains a measure
0.287	Contains link	0.071	Is reply-tweet	0.011	No. hashtags
0.130	Formality score	0.060	Avg. word length	0.008	No. of mentions
0.127	Num. proper names	0.042	% of correct spell.	0.007	Contains emoticons
0.113	Max. proper names	0.041	Hour of the day	0.007	Contains nums
0.111	Tweet length	0.041	Hashtag reputation	0.005	No. quest. marks
0.089	No. named entities	0.034	RT source reput.	0.003	No. capital. words
0.087	% of stopwords	0.023	Max. repeated chars.	0.001	Is re-tweet
0.083	Max. word length	0.023	Uniqueness score	0.000	Max. capital. words
0.081	Has first-person	0.019	No. excl. marks		

the features, however, do not provide as useful characterization. We observe that spelling and punctuation are not particularly strong indicators, which may be due to the informal language (e.g., short forms and abbreviations) often used in Twitter. Also, the number of '@username' mentions is not a strong indicator. We observe that while some tweets mentioning many users generally have lower quality (e.g. private conversations between a group of users), many high-quality tweets also mention users, such as names of public figures or organizations. Based on IG, the 'Is re-tweet' feature is not a strong indicator of high-quality tweets, aligning with our observation in Section 6.1 that even re-tweets may contain low-quality or noisy messages. In contrast, 'RT source reputation' proves to be a clearly stronger indicator, leveraging the reputation of re-tweeted users.

An overview of feature sets used for experiments is presented in Table 5.

Filtering Results. We evaluate the accuracy of a SVM classifier on different feature sets for the filtering of high-quality, as well as low-quality tweets. The results are presented in Table 6.

According to the AUC results in Table 6, 'Subset.SVM' performs the best among all the feature sets, achieving the highest recall in high-quality filtering, also achieving the highest precision in low-quality filtering. Furthermore, the link-based features ('C4.Links') also archive high AUC ($AUC = 0.84$), especially the subset that contains only reputation-based features ('Subset.Reput', $AUC = 0.841$). The two sets ('C4.Links' with 8 features only, and 'Subset.Reputation' with 3 features only) outperform the 'TFIDF' method (with 3322 features, corresponding to term n-grams) and also the 'Link-only' feature used in [2]. Link-based features are also useful in filtering out high-quality tweets: among the

Table 5. Description of feature sets used in experiments

Feature set	*#Ftr's*	*Description*
Text (TFIDF)	3322	TFIDF represent. of term n-grams up to length 5. Baseline.
Link only	1	Single feature - presence of a hyperlink. Method used in [2].
C1.Spell	6	Punctuation and spelling features
C2.Comp	8	Syntactic and semantic complexity features
C3.Gram	5	Grammaticality features
C4.Links	8	Link-based features
C5.Time	2	Timestamp features
Subset.Cont	19	All content-based features (C1 - C3).
Subset.Reput	3	Reputation score features (DRS, RRS, HRS)
Subset.SVM	15	Features selected by greedy attribute selection (see Table 3)
All features	29	All content and link-based features
All ftr's + Text	3351	All content, link and TF-IDF features

Table 6. Precision (P), Recall (R) and Area Under the ROC Curve (AUC) results for the task of finding high-quality and low-quality tweets using different feature sets

	High-Quality		*Low-Quality*				*High-Qual.*		*Low-Qual.*		
Features	*P*	*R*	*P*	*R*	*AUC*	*Features*	*P*	*R*	*P*	*R*	*AUC*
Text (TFIDF)	**0.862**	0.665	0.885	0.96	0.813	Link only	0.798	0.702	0.894	0.934	0.818
Subset.Cont	0.721	0.61	0.863	0.913	0.762	C1.Spell	0.5	0.004	0.73	0.999	0.501
Subset.Reput	0.812	0.746	0.909	0.936	**0.841**	C2.Comp	0.628	0.165	0.757	0.964	0.564
Subset.SVM	0.715	**0.758**	**0.912**	0.936	**0.847**	C3.Gram	0.648	0.472	0.822	0.905	0.688
All features	0.815	0.66	0.882	0.944	0.802	C4.Links	**0.82**	0.74	0.907	0.94	**0.84**
All ftr's+text	0.739	**0.775**	**0.915**	0.899	0.837	C5.Time	0	0	0.729	1	0.5

5 feature categories (C1 - C5), 'C4.Links' yields the best precision for high-quality tweets. However, it does not yield the best recall, because we find that quite a large portion of tweets in our dataset do not contain hyperlinks (68.9%) or hashtags (85.9%), and thus 'C4.Links' features cannot be directly applied to them. Finally, 'Subset.Cont' yields relatively high precision on low-quality tweets, showing that Content-based features are fairly useful in filtering out low-quality tweets.

We observe that 'TFIDF' yields high precision for high-quality tweets, because it employs a large number of features in the classification. However, a comparison of the training time and storage space requirements (shown in Figure 6) reveals that 'TFIDF' consumes the largest amount of training time and space due to the large number of features (i.e., 3322 features). Overall, the optimal feature subset 'Subset.SVM' not only yields better overall results, but also requires less training time and space compared to the 'TFIDF' representation.

6.3 Ranking Evaluation

Experiment Methodology. To evaluate our ranking method, we manually prepare 30 single-word test queries in the ranking evaluation. The queries are

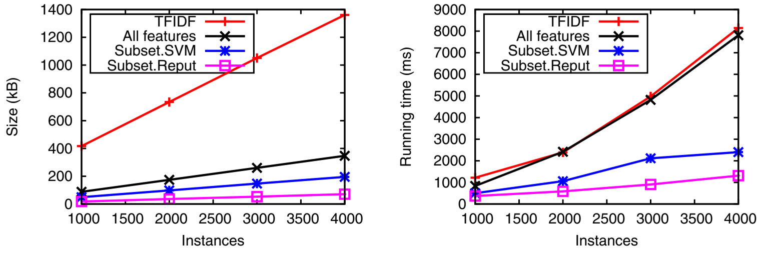


Fig. 6. Storage cost (left) and training time cost (right) of different feature sets

Table 7. Ranking accuracy in terms of NDCG@N and Mean Average Precision (MAP)

	NDCG@N						NDCG@N				
Features	*1*	*2*	*5*	*10*	*MAP*	*Features*	*1*	*2*	*5*	*10*	*MAP*
Link only	0.067	0.111	0.22	0.324	0.398	C1.Spell	0.511	0.456	0.466	0.51	0.5
Subset.Cont	0.622	0.644	0.653	0.651	0.55	C2.Comp	0.8	0.756	0.639	0.603	0.536
Subset.Reput	0.822	**0.777**	**0.777**	0.764	**0.661**	C3.Gram	0.622	0.6	0.612	0.6	0.513
Subset.SVM	**0.867**	0.767	**0.778**	**0.769**	0.653	C4.Links	0.733	0.656	0.687	0.711	0.639
All features	0.733	0.733	0.763	0.753	0.637	C5.Time	0.156	0.267	0.282	0.346	0.377

randomly selected from 5 different categories: News, Politics, Technology, Business and Entertainment, to avoid any topical bias. For each query, a set of labeled tweets containing the query term is retrieved. A total of 1,834 tweets are retrieved for the 30 test queries. We divide the 1,834 tweets into two sets, one set for the training and the other set for the Rank SVM testing. Basically, the tweets retrieved from 15 test queries are used for the training, while the tweets from the remaining 15 test queries are used for the testing. In the ranking evaluation, we use Normalized Discounted Cumulative Gain ($NDCG$) and Mean Average Precision (MAP), which are standard metrics for evaluating ranking accuracy.

Ranking Results. We evaluate the ranking model with different features as shown in Table 7. We observe that 'Subset.Reput' (the set of reputation-based features) and 'Subset.SVM' achieve the overall best results ($MAP = 0.661$ and 0.653), significantly outperforming the 'Link only' feature used in [2]. Furthermore, among the 5 sets of features (C1 - C5) proposed in Section 4.2, link-based features achieve the best results. This aligns with our observations in Section 6.2 that link-based features (especially the three reputation-based features) are useful for identifying high-quality tweets.

7 Conclusion

In this paper, we study the problem of finding high quality content in Twitter. We formulate the criteria of quality tweets and tackle the filtering and tweet ranking problems. The quality of a tweet is modelled using a set of features based on the tweet's content, as well as links to websites, hashtags and users. Our proposed link-based features are able to boost the filtering and ranking performance, indicating that the implicit "reputation" of a web domain, hashag or re-tweeted user is highly useful in the filtering and ranking tasks. In our experiments, the optimal feature subset that includes link-based features achieves the best overall classification and ranking accuracy.

Although we focus on Twitter in this work, the results are potentially useful in the contexts of other social networks and microblogging services. For future work, we plan to consider different types of queries in Twitter (e.g. hot topic queries, movie reviews, highly factual seeking queries) and study the importance of tweet features for filtering and ranking in these different scenarios.

Acknowledgments. This work is partially supported by Hong Kong RGC GRF project grant 618509.

References

1. TwitterEngineering: 200 million tweets per day, http://blog.twitter.com/2011/06/200-million-tweets-per-day.html
2. Alonso, O., Carson, C., Gerster, D., Ji, X., Nabar, S.: Detecting Uninteresting Content in Text Streams. In: Proc. of SIGIR CSE Workshop (2010)
3. Java, A., Song, X., Finin, T., Tseng, B.: Why We Twitter: Understanding Microblogging Usage and Communities. In: Zhang, H., Spiliopoulou, M., Mobasher, B., Giles, C.L., McCallum, A., Nasraoui, O., Srivastava, J., Yen, J. (eds.) WebKDD 2007. LNCS, vol. 5439, pp. 118–138. Springer, Heidelberg (2009)
4. Zhao, D., Rosson, M.B.: How and why people twitter: the role that micro-blogging plays in informal communication at work. In: Proc. of GROUP (2009)
5. Sriram, B., Fuhry, D., Demir, E., Ferhatosmanoglu, H., Demirbas, M.: Short text classification in twitter to improve information filtering. In: Proc. of SIGIR Conference (2010)
6. Ramage, D., Dumais, S.T., Liebling, D.J.: Characterizing microblogs with topic models. In: Proc. of ICWSM Conference (2010)
7. Lauw, H., Ntoulas, A., Kenthapadi, K.: Estimating the quality of postings in the real-time web. In: Proc. of SSM Conference (2010)
8. Nagmoti, R., Teredesai, A., De Cock, M.: Ranking approaches for microblog search. In: Proc. of WI-IAT Conference (2010)
9. Trifan, M., Ionescu, D.: A new search method for ranking short text messages using semantic features and cluster coherence. In: Proc. of ICCC-CONTI (2010)
10. Sarma, A.D., Sarma, A. D., Gollapudi, S., Panigrahy, R.: Ranking mechanisms in twitter-like forums. In: Proc. of WSDM Conference (2010)
11. Duan, Y., Jiang, L., Qin, T., Zhou, M., Shum, H.-Y.: An empirical study on learning to rank of tweets. In: Proc. of COLING Conference (2010)
12. Grove, J.V.: Twitter attempts to personalize 1.6 billion search queries per day, http://mashable.com/2011/06/01/twitter-search-queries/
13. Toutanova, K., Klein, D., Manning, C., Singer, Y.: Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proc. of HLT-NAACL (2003)

14. Lahiri, S., Mitra, P., Lu, X.: Informality Judgment at Sentence Level and Experiments with Formality Score. In: Gelbukh, A. (ed.) CICLing 2011, Part II. LNCS, vol. 6609, pp. 446–457. Springer, Heidelberg (2011)
15. Finkel, J.R., Grenager, T., Manning, C.: Incorporating non-local information into information extraction systems by gibbs sampling. In: Proc. of ACL (2005)
16. Joachims, T.: Optimizing search engines using clickthrough data. In: Proc. of ACM SIGKDD Conference (2002)
17. Cheng, Z., Caverlee, J., Lee, K.: You are where you tweet: a content-based approach to geo-locating twitter users. In: Proc. of CIKM Conference (2010)
18. Kohavi, R., John, G.H.: Wrappers for feature subset selection. Artificial Intelligence 97, 273–324 (1997)

HotDigg: Finding Recent Hot Topics from Digg

Younghoon Kim and Kyuseok Shim

Seoul National University, Seoul, Korea
{yhkim,shim}@kdd.snu.ac.kr

Abstract. The popular news aggregator called Digg is a social news service that lets people share new articles or blog postings in web pages with other users and vote thumbs up and thumbs down on the shared contents. Digg itself provides only the functionality to search the articles for the topics provided by users using manually tagged keywords. Helping users to find the most interesting Digg articles with the current hot topics will be very useful, but it is not an easy task to classify the articles according to their topics and discover the articles with the hot topics quickly.

In this paper, we propose HotDigg, a recommendation system to provide the articles with hot topics in Digg using a novel probabilistic generative model suitable for representing the activities in Digg service. We next propose an EM algorithm to learn the parameters of our probabilistic model. Our performance study with real-life data from Digg confirms the effectiveness of HotDigg by showing that the articles with current hot topics are recommended.

1 Introduction

The popular news aggregator called Digg [6] is a social news service that allows us to share our articles in web pages and vote the likeness of the articles posted by others. When a Digg user finds an article that he wants to share, he can submit the URL with a brief description of the article to Digg web site and let people vote thumbs up or thumbs down, called *digging* or *burying* respectively, for the submission. The submissions with a lot of "diggs" (i.e. community votes of thumbs up) but also new submissions are displayed in reverse chronological order at Digg web site with their digg scores where the digg score of an article is the number of diggings subtracted by the number of buryings.

Generally, users would like to see the articles of the hot topics. However, Digg itself provides only the functionality to search the articles using manually tagged keywords for the topics provided by users and display the articles in the search result in the decreasing order of voting scores. Helping users to find the most interesting Digg articles with the current hot topics will be very useful, but it is not an easy task to classify the articles according to their topics and discover the articles with the hot topics quickly.

In our paper, we propose HotDigg, a system for displaying the articles with hot topics in Digg using probabilistic modeling. Due to the characteristics of

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 414–427, 2012.
© Springer-Verlag Berlin Heidelberg 2012

Digg that users can post and vote the articles with their interesting topics, the articles with each similar topic generally tend to get similar digg scores. Thus, we develop a generative model from a unifying viewpoint that each article is produced by a topic mixture model and its votes are determined by its topic.

Our probabilistic model is a generalization of the probabilistic latent semantic indexing (PLSI) in [7]. If we blindly apply the model of PLSI to Digg, we cannot utilize the characteristics that the articles with the same topic obtain similar digg scores. To take advantage of the characteristics, we first propose a new probabilistic model by assuming that the dig scores of the articles on each topic follow a Gaussian distribution. We next develop an Expectation-Maximization(EM) algorithm to learn the parameters of our model by maximizing the log-likelihood of expectation. The contributions of this paper are as follows:

- We develop a novel probabilistic generative model which is suitable for representing the activities in Digg service.
- We propose an EM algorithm to learn the parameters of our probabilistic model.
- Our performance study with real-life data from Digg confirms the effectiveness of HotDigg by showing that the articles with current hot topics are recommended.

2 Related Work

In this section, we discuss the model-based emerging topic detection algorithms [2,7,10,9,19,16] and next describe the recent works on Digg service [11,12].

Emerging topic detection refers to the automatic techniques for finding topically related contents in the streams of data such as newswire and broadcast. One approach to identify emerging topics in document streams is to cluster new articles so that the articles in each cluster contain similar words. Extensive research in the area of document clustering and topic modeling with Latent Dirichlet Allocation [2], Probabilistic Latent Semantic Index (PSLI) [7], and Non-negative Matrix Factorizations [10] can be utilize for finding topics. However, these clustering techniques alone cannot help to determine whether the clusters found are related to the emerging topics or popular topics.

In [9], given a collection of text data with published time, an algorithm is proposed to identify emerging topics. However, this algorithm does not identify popular topics but focuses on discovery of the emerging topics only. In [19], to detect topics from a stream of text data in real time, an emerging topic detection algorithm which focuses on tracking the changes of topics through time is developed. However, it cannot estimate the popularity of the detected emerging topics. In [16], an algorithm to analyze the life cycles of topics using a modified PLSI model is developed. However, it cannot identify the popular topics utilizing vote scores such as the digg score of Digg service.

In [11,12], by using the voting scores of the submitted contents in the web sites such as Digg and Del.icio.us (a social network service for sharing bookmarks [3]), they shows the characteristics that popular topics are spread over the social networks. However, they do not provide the algorithm to find the popular topics.

3 Preliminaries

We first provide the definitions to define our problem and present the problem formulation.

3.1 Problem Formulation

Observed Data: Let $D = \{d_1, ..., d_n\}$ be a collection of the articles submitted to Digg service. Each article $d_i \in D$ is a bag of words which is generated by deleting the stop-words from its original Digg article. Digg users can vote thumbs up or thumbs down, called *digging* or *burying* respectively, for each Digg article. Every Digg article $d_i \in D$ has a *digg score* s_i of an integer which is the number of diggings subtracted by the number of buryings voted for d_i. Let $W = \{w_1, ..., w_m\}$ be a set of words occurring in a Digg article $d_i \in D$. We define $n(d_i, w)$ to denote the number of occurrences of $w \in W$ in a Digg article $d_i \in D$.

Unobserved Topics: We assume that there exist t major topics denoted by $Z = \{z_1, z_2, ..., z_t\}$ in a collection D of Digg articles. The z_k is also used as a *hidden variable* to represent one of the t topics in our probabilistic model. Note that we do not know in advance the actual keywords representing each topic z_k in Z. For each article d_i in D, we assume that d_i belongs to each topic z_k with the probability $p(z{=}z_k|d{=}d_i)$. For each article d_i in D, we define the topical score of d_i for a topic z_k in Z, denoted by $s_t(d_i, z_k)$, as $p(z{=}z_k|d{=}d_i) \cdot s_i$ which is the digg score of d_i weighted by $p(z{=}z_k|d{=}d_i)$. Then, for each topic z_k, we define the topic score τ_k as the weighted average score of the $s_t(d_i, z_k)$ with every article d_i in D and compute τ_k as follows:

$$\tau_k = \frac{\sum_{d_i \in D} p(z{=}z_k|d{=}d_i) \cdot s_i}{\sum_{d_i \in D} p(z{=}z_k|d{=}d_i)} = \frac{\sum_{d_i \in D} s_t(d_i, z_k)}{\sum_{d_i \in D} p(z{=}z_k|d{=}d_i}. \tag{1}$$

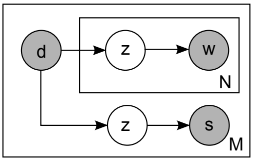
Finally, for each digg article $d_i \in D$ and a subset of topics $S \subseteq Z$, we define $s_T(d_i, S)$ as the topical score of d_i for S which is the sum of the topical score $p(z{=}z_k|d{=}d_i) \cdot s_i$ with every topic $z_k \in S$, calculated by

$$s_T(d_i, S) = \sum_{z_k \in S} s_t(d_i, z_k). \tag{2}$$

Problem Definition: Assume that we have a collection D of Digg articles and the digg score for each article in D. Given k_1, k_2 and t, our problem is to partition the Digg articles in D into t clusters according to their topics in order to

Article Id	Digg Score	Words
d1	1	Texas, wildfire, flame, Tuesday
d2	3	Texas, wildfire, Tuesday
d3	2	flame, Yahoo
d4	10	Texas, Yahoo, CEO, company
d5	12	Tuesday, Yahoo, CEO

(a) An example of digg articles and their scores



(b) An graphical model

Fig. 1. An example of digg articles and our topic model

find the set S of the top-k_1 topics with the highest topic scores (i.e., τ_k) among t topics and then we select the top-k_2 articles with the highest topical scores (i.e., $s_T(d', S)$) among all articles in D.

Example 1. Consider a set of Digg articles $D = \{d_1, ..., d_5\}$ with their dig scores shown in Figure 1(a). The Digg articles are presented after eliminating stop-words. Suppose that we want to show 2 articles with the top-1 popular topics (i.e., k_1=1 and k_2=2). In the examples, there exist 2 topics which are related to 'Texas wildfire' and 'Yahoo's CEO'. Since most of words in the articles d_1 and d_2 represent the topic 'Texas wildfire', d_1 and d_2 should be clustered together. However, it is not clear to determine the topic of d_3 because the half of the words appearing in d_3 is related to 'Texas wildfire' and the other word is related to 'Yahoo's CEO'. Assuming that the articles on the same topic get similar popularity and similar digg scores, we can assign d_3 into the cluster of the topic 'Texas wildfire'. Furthermore, to find the top-2 articles with the top-1 popular topic, since the digg scores of d_4 and d_5 have the two highest probabilities for the topic 'Yahoo's CEO' and the topic score of the topic 'Yahoo's CEO' is the highest, we select the articles d_4 and d_5. ∎

4 Our Generative Model for Digg

Many previous works have shown the effectiveness of mixture models in clustering text collections by hidden topics using conditional probability distributions [7,2,16,15,17,13]. However, since the traditional mixture models do not consider any score data such as digg scores to measure the popularity of each document, we next propose a new topical model for Digg by utilizing not only the textual contents but also the digg scores.

4.1 Our Probabilistic Model

To model both the topics and the scores of Digg articles, we use a mixture of conditional probability distributions, as the PLSI model in [7] does, but generalize the model structure to include the digg scores. In our generative model, we assume that each word in a Digg article is chosen for a topic z_k in Z similar to the model of PLSI. Furthermore, we assume that for each Digg article, a digg score value is also selected for the topic of the article following to a Gaussian distribution whose mean and variance depend on the topic only. Correspondingly, we introduce the following three conditional probability density functions (PDFs):

(1) $p(z|d{=}d_i)$ is the conditional PDF with which we select the topic $z \in Z$ for the given digg article $d_i \in D$.
(2) $p(w|z{=}z_k)$ is the conditional PDF with which the word $w \in W$ is selected depending on the given topic $z_k \in Z$.
(3) $p(s|z{=}z_k)$ is the conditional PDF such that, given the topic $z_k \in Z$, a digg score value $s_i \in Z$ is chosen by following the probability $p(s{=}s_i|z{=}z_j)$. We assume that $p(s|z{=}z_j)$ is a Gaussian function.

As long as it is clear from the context, we will denote $p(z|d{=}d_i)$, $p(w|z{=}z_k)$ and $p(s|z{=}z_k)$ by $p(z|d_i)$, $p(w|z_k)$ and $p(s|z_k)$ respectively for the sake of simple representation.

The Topic Selection Model: A topic z_k is selected with the dependency on each Digg article. We introduce the conditional PDF of $p(z_k|d_i)$ which represents the relevance of a Digg article $d_i \in D$ to a topic $z_k \in Z$. Note that $\sum_{z_k \in Z} p(z{=}z_k|d_i) = 1$ holds for every Digg article $d_i \in D$.

The Word Selection Model: Given a topic $z_k \in Z$, a word $w_j \in W$ is chosen with the probability $p(w{=}w_j|z_k)$ depending on z_k only. Obviously, we have $\sum_{w_j \in W} p(w{=}w_j|z_k) = 1$ for every topic $z_j \in Z$. Semantically, the probability distribution represents the relevancy of words to describe a given topic.

The Score Selection Model: Given a topic $z_k \in Z$, a score value $s_i \in Z$ is generated with the probability $p(s{=}s_i|z_k)$ depending on z_k only. For the PDF of $p(s{=}s_i|z_k)$, we use a Gaussian distribution $N(s_i; \tau_k, \sigma_k^2)$, where τ_k and σ_k^2 are the mean and the variance of the distribution for the topic z_k.

In Figure 1(b), we show the graphical representation of our proposed mixture model. Our generative model represents two independent generative processes. The first process illustrates the writings with hidden topics by the authors of Digg articles. The second process represents how the digg scores are produced with hidden topics by the Digg users.

Digg Article Generation: In the first process, each Digg article is produced by repeating the following steps while choosing the words stochastically:

1. We first select a topic z_k for the Digg article d_i by following the conditional probability density function $p(z|d_i)$.
2. With the topic z_k chosen in the above, a word w_j is selected by following the conditional probability density function $p(w|z_k)$.

We repeat the above two steps as many times as the number of words in d_i for every article $d_i \in D$.

Digg Score Determination: To the process the selection of the words independently, a digg score is also determined for each Digg article by performing the following steps:

1. We first choose a topic z_k for the Digg article d_i by following the conditional probability density function $p(z|d_i)$.
2. With the topic z_k chosen in the above, a digg score s_i is determined by following the Gaussian probability distribution of $N(s;\tau_k,\sigma_k^2)$.

We now formally present our probabilistic model and its maximum likelihood estimate to compute the conditional probability distributions $p(w|z_k)$s, $p(z|d_i)$s and $N(s;\tau_k,\sigma_k^2)$s for a given Digg data.

4.2 The Likelihood of a Digg Article Collection

Let D be a collection of Digg articles. For each Digg article $d_i \in D$, we have a tuple $\langle w(d_i), s_i\rangle$ where $w(d_i)$ and s_i are a bag of words occurring at d_i and a digg score of d_i respectively. With the probability $p(w(d_i), s_i|d_i)$, the likelihood of the collection D according to our generative model is

$$\mathbb{L} = \prod_{d_i \in D} p(w(d_i), s_i|d_i).$$

Since the words appearing in d_i and the digg score of d_i are sampled independently, we have $p(w(d_i), s_i|d_i) = p(w(d_i)|d_i) \cdot p(s{=}s_i|d_i)$. Furthermore, since each word in a digg article is sampled identically and independently, we can write the likelihood as follows

$$\mathbb{L} = \prod_{d_i \in D} \left[p(s{=}s_i|d_i) \prod_{w_j \in W} p(w{=}w_i|d_i)^{n(d_i,w_j)} \right],$$

where $n(d_i, w_j)$ denotes the number of appearances of the word w_j in the digg article d_i.

By marginalization with the random variable z for topics, $p(w{=}w_j|d_i)$ becomes $\sum_{z_k \in Z} p(w{=}w_j|z_k) \cdot\ p(z{=}z_k|d_i)$. Similarly, we have $p(s{=}s_i|d_i) = \sum_{z_k \in Z}$ $p(s{=}s_i|z_k) \cdot\ p(z{=}z_k|d_i)$. Then, by using $p(s{=}s_i|z_k) = N(s_i;\tau_k,\sigma_k^2)$, we obtain

$$\mathbb{L} = \prod_{d_i \in D} \left[\sum_{z_k \in Z} N(s_i;\tau_k,\sigma_k^2) p(z{=}z_k|d_i) \right] \cdot \left[\prod_{w_j \in W} \left\{ \sum_{z_k \in Z} p(w{=}w_j|z_k) p(z{=}z_k|d_i) \right\}^{n(d_i,w_j)} \right]. \tag{3}$$

4.3 The Maximum Likelihood Estimate

Assume that the observed data is generated from our generative model. Let Θ denote the initially unknown four parameters of our model, which are $p(w|z)$, $p(z|d)$, τ and σ^2. We wish to find Θ such that the likelihood $\mathbb{L}$ in Equation (3) is maximized. This is known as the Maximum Likelihood (ML) estimation [18] for computing Θ. In order to estimate Θ, it is typical to introduce the log-likelihood function defined as

$$\log \mathbb{L} = \sum_{d_i \in D} \left[\log \sum_{z_k \in Z} N(s_i; \tau_k, \sigma_k^2) p(z{=}z_k|d_i) + \sum_{w_j \in W} n(d_i, w_j) \cdot \log \sum_{z_k \in Z} p(w{=}w_j|z_k) p(z{=}z_k|d_i) \right] . \tag{4}$$

The likelihood function is considered to be a function of the parameter Θ for a Digg collection D. Since $\log \mathbb{L}$ is a strictly increasing function, the value of Θ which maximizes log-likelihood of $\log \mathbb{L}$ also maximizes the likelihood $\mathbb{L}$ [20]. Since the parameters $p(z|d)$ and $p(w|z)$ are probability values, we have the following constraints:

$$\sum_{z_k \in Z} p(z{=}z_k|d_i) = 1 \text{ for every } d_i \in D, \sum_{w_j \in W} p(w{=}w_j|z_k) = 1 \text{ for every } z_k \in Z$$

Thus, we have to calculate the model parameters Θ with maximizing the log-likelihood $\log \mathbb{L}$ in Equation (4) with the above constraints.

5 Estimation of Model Parameters

Without any prior knowledge, we can apply the maximum likelihood estimator to compute all the parameters by applying the Expectation-Maximization (EM) algorithm [4]. An EM algorithm performs the iterations with two steps of an expectation step (E-step) and a maximization step (M-step). In E-step, the probability expectation of the hidden variables is computed by using the current estimate of parameters, and in M-step, the parameters maximizing the log-likelihood is calculated by utilizing the expectation computed in E-step. The parameters estimated in M-step are then used in E-step of the next iteration.

The E-Step: This step calculates the expectation of the hidden variables. Here, the hidden variable is (1) the topic z_k which is chosen for generating each word w_j for d_i, (2) the topic z_t which is selected for determining the digg score s_i. Let $p(z{=}z_k|d_i, w_j)$ be the expected probability that a word w_j is generated from the topic z_k in the Digg article d_i. Similarly, let $p(z{=}z_t|d_i, s_i)$ be the expected probability that a topic z_t is selected for given a Digg article d_i and its digg score. The formulas for computing these probability expectations are presented in Figure 2.

The M-Step: In order to find the parameters Θ maximizing Equation (4), we apply the method of Lagrange multipliers [1]. We list the obtained formulas for the M-Step of our EM algorithm in Figure 2.

We iterate E-Step and M-Step until we obtain the convergence of the log-likelihood in Equation (4). Since our EM algorithm only guarantees to find a local maximum of the likelihood, we perform multiple trials and choose the best local maximum among the local optima found.

E-step:

$$p(z{=}z_k|d_i,w_j) = \frac{p(w{=}w_j|z_k)p(z{=}z_k|d_i)}{\sum_{z'\in Z} p(w{=}w_j|z')p(z{=}z'|d_i)} \tag{5}$$

$$p(z{=}z_t|d_i,s_i) = \frac{N(s_i;\tau_t,\sigma_t^2)p(z{=}z_t|d_i)}{\sum_{z_{t'}\in Z} N(s_i;\tau_{t'},\sigma_{t'}^2)p(z{=}z_{t'}|d_i)} \tag{6}$$

M-step:

$$p(w{=}w_j|z_k) = \frac{\sum_{d_i\in D} n(d_i,w_j)\cdot p(z{=}z_k|d_i,w_j)}{\sum_{w'\in W}\sum_{d_i\in D} n(d_i,w')\cdot p(z{=}z_k|d_i,w')} \tag{7}$$

$$p(z{=}z_k|d_i) = \frac{\sum_{w_j\in W} n(d_i,w_j)\cdot p(z{=}z_k|d_i,w_j) + p(z{=}z_k|d_i,s_i)}{\sum_{z'\in Z}[\sum_{w_j\in W} n(d_i,w_j)\cdot p(z{=}z'|d_i,w_j) + p(z{=}z'|d_i,s_i)]} \tag{8}$$

$$\tau_k = \frac{\sum_{d_i\in D} s_i\cdot p(z{=}z_k|d_i,s_i)}{\sum_{d_i\in D} p(z{=}z_k|d_i,s_i)} \tag{9}$$

$$\sigma_k^2 = \frac{\sum_{d_i\in D}(s_i-\tau_k)^2\cdot p(z{=}z_k|d_i,s_i)}{\sum_{d_i\in D} p(z{=}z_k|d_i,s_i)} \tag{10}$$

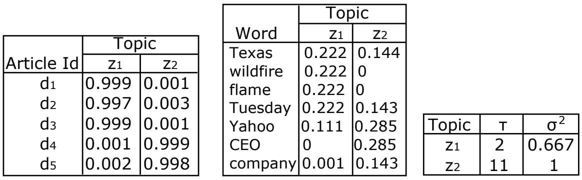
Fig. 2. The formulas for E-step and M-step

Example 2. Consider a set D of Digg articles shown in Figure 1(a). Since the data set contains two topics which are 'Texas wildfire' and 'Yahoo's CEO', we set the number of topics t to 2 in this example. In the E-step of the first iteration, for every Digg article $d_i \in D$, every topic $z_k \in Z$ and every word $w_j \in W$, the conditional probabilities $p(z{=}z_k|d_i)$ and $p(w{=}w_j|z_k)$ are initially set to 0.5 and 0.142 respectively to satisfy $\sum_{z_k\in Z} p(z{=}z_k|d) = 1$ and $\sum_{w_j\in W} p(w{=}w_j|z) = 1$. Furthermore, we set $\tau_{z_1} = 2$, $\sigma_{z_1}^2 = 4$, $\tau_{z_2} = 8$ and $\sigma_{z_2}^2 = 4$.

In the E-step of the first iteration, we compute $p(z{=}z_k|d_i,w_j)$ and $p(z{=}z_k|d_i,s_i)$ with the previously initialized model parameters. Let us compute $p(z{=}z_1|d_1,s_1)$ and $p(z{=}z_2|d_1,s_1)$. When $N(1;\tau{=}2,\sigma^2{=}2) = 0.176$ and $N(1;\tau{=}8,\sigma^2{=}2) = 0.0004$, $p(z{=}z_1|d_1,s_1)$ and $p(z{=}z_2|d_1,s_1)$ become 0.998 and 0.002 respectively. Then, in the M-step, the model parameters are computed according to Equations (5)-(10). The log-likelihood $\mathbb{L}$ after the first iteration becomes -51.32. After repeating E-step and M-step until the log-likelihood converges to -42.86, we obtain the conditional probability distributions as shown in Figure 3.

As we can expect from Digg articles in Figure 1(a), the Digg articles d_1, d_2 and d_3 have high probabilities for the topic $z{=}1$. In contrast, d_4 and d_5 have high probabilities for the topic $z{=}2$. From the values of $p(w|z)$ in Figure 3(b), we can see that that words 'Texas', 'wildfire' and 'Tuesday' occur more frequently for the topic $z{=}1$ while 'Yahoo', 'CEO' and 'company' tend to appear more for the topic $z{=}2$. Furthermore, the values of τ_z are listed in Figure 3(c). ∎

Time and Space Complexities: In a single E-step, the time complexity of computing all $p(z{=}z_k|d_i,w_j)$s for every pair of $d_i \in D$ and $w_j \in W$ becomes $O(|Z|\cdot|D|\cdot|W|)$. Since the computation of $p(z{=}z_t|d_i,s_i)$s takes only $O(|Z|\cdot|D|)$ time, the time complexity of an E-step becomes $O(|Z|\cdot|D|\cdot|W|)$.



Article Id	Topic	
	z_1	z_2
d_1	0.999	0.001
d_2	0.997	0.003
d_3	0.999	0.001
d_4	0.001	0.999
d_5	0.002	0.998

(a) p(z|d)

Word	Topic	
	z_1	z_2
Texas	0.222	0.144
wildfire	0.222	0
flame	0.222	0
Tuesday	0.222	0.143
Yahoo	0.111	0.285
CEO	0	0.285
company	0.001	0.143

(b) p(w|z)

Topic	τ	σ^2
z_1	2	0.667
z_2	11	1

(c) τ and σ^2

Fig. 3. The resulting conditional probability distributions

To compute all $p(w{=}w_j|z_k)$s, $p(z{=}z_k|d_i)$s, τ_{z_k}s and $\sigma^2_{z_k}$s in Equations (7)-(10), we simply scan every Digg article d_i and update those parameters holding aggregated values of $p(z{=}z_k|d_i, w_j)$ for each occurrence of every word w_j. Thus, the time complexity of an M-step is $O(D_W \cdot |Z|)$ where $D_W = \sum_{d_i \in D} \sum_{w_j \in W} n(d_i, w_j)$. Note that D_W is the appearance count of all words in all Digg articles. The overall time complexity of a single iteration in our EM algorithm becomes $O(|Z| \cdot |D| \cdot |W| + D_W \cdot |Z|)$.

To maintain all values of $p(z{=}z_k|d_i, w_j)$ and $p(z{=}z_s|d_i, s_i)$, we need $O(|Z| \cdot |D| \cdot |W|)$ space of main memory. For $p(w{=}w_j|z_k)$, $p(z{=}z_k|d_i)$, τ_{z_k} and $\sigma^2_{z_k}$, we need $O((|W| + |D|) \cdot |Z|)$ space. Thus, the total space complexity of our EM algorithm becomes $O(|Z| \cdot |D| \cdot |W|)$.

6 Finding Top-k_1 Topics and Top-k_2 Digg articles

After the parameters in our model are estimated using the EM algorithm presented in the previous section, we find the top-k_1 popular topics and the top-k_2 Digg articles which not only have high digg scores but also are related to the popular topics utilizing the estimated model parameters. According to our problem definition, we first find the top-k_1 topics S with the k_1 largest topic scores (i.e., τ_k) estimated by our EM algorithm. Then, using the probabilities $p(z{=}z_k|d_i)$ computed by our EM algorithm as model parameters, we compute the topical scores $s_T(d_i, S)$ in Equation (2) for every Digg article $d_i \in D$. Finally, we select the top-k_2 Digg articles with the k_2 largest topical scores among all $d_i \in D$.

7 Experiments

We empirically evaluated the performance of our proposed algorithm *HotDigg*. All experiments reported in this section were performed on the machines with Intel(R) Core(TM)2 Duo 2.66GHz and 2GB of main memory running Linux. All algorithms were implemented using Java Compiler of version 1.6.

7.1 Implemented Algorithms

For our experiments, we used the following algorithms.

- **HotDigg:** This is the implementation of our proposed algorithm.
- **NAIVE:** This is the implementation of the naive algorithm which we simply find the k_2 Digg articles with the k_2 largest digg scores. Note that we cannot find the popular topics using this algorithm.
- **PLSI:** It is the implementation of algorithm using PLSI algorithm in [7]. Since we can also obtain the probability $p(z{=}z_k|d_i)$ that a Digg article d_i belongs to a topic z_k using PLSI algorithm, we can compute the topic score τ_k in Equation (1) and select the top-k_1 topics S using the topic scores. Furthermore, using the probability $p(z{=}z_k|d_i)$, we can calculate the topical score $s_T(d_i, S)$ in Equation (2) for every Digg article $d_i \in D$ and choose the top-k_2 Digg articles with the k_2 largest topical scores.

7.2 Data Sets

For experimental study, we evaluate the algorithms on real-life data. We downloaded 30,000 Digg articles at September 9, 2011 using the Digg API [8]. We removed the stop words appearing in more than 80% of all articles. Furthermore, we also deleted the words occurring in less than 3 articles since such words do not provide any clue for topical clustering. For all words in Digg articles, Lovins stemmer [14] was used to stem the words. We call this data as ORG-DATA.

To test the accuracy of the hot topics found by our algorithm, we selected 4 topics appearing in the ORG-DATA and we found that these topics were related to “Texas wildfire”, “Yahoo’s CEO”, “Politics” and “Facebook cheats” respectively. Then, for each i-th selected topic, we manually extracted 20 articles from the ORG-DATA which address the selected topic and increased the digg scores of the 20 articles by multiplying 10. Then, we generated the test data set called TEST-DATA(i) by replacing the modified articles with the original articles in ORG-DATA. Since we selected 4 topics, we have four data sets which are TEST-DATA(1), TEST-DATA(2), TEST-DATA(3) and TEST-DATA(4).

The digg scores in ORG-DATA follow Zipf’s distribution and thus have very wide range. Since our model in Section 4 assumes that the digg scores in a single topic follow a Gaussian distribution, biased digg scores can affect the accuracy of our model. Thus, we use the logarithm of digg scores instead.

7.3 The Result of Topical Clustering

We first conducted experiments with our proposed *HotDigg* algorithm using ORG-DATA to find the hot topics. We set number of topics t to 30. We found that the top-4 topics among the hidden topics were related to “Texas Wildfire”, “Yahoo’s CEO”, “Politics” and “Facebook Cheats” respectively.

In Figure 4, we show 2 Digg articles with the 2 largest topical scores, (i.e., $p(z_k|d_i) \cdot s_i$) among all Digg articles in D for each topic of the selected 4 topics. The result confirms that our topic model clusters the digg articles appropriately according to their topics. Moreover, we can confirm that the Digg articles in a single topic generally have the similar digg scores as we assumed for our model.

	Votes	Contents
Topic1	230	Texas wildfires destroy more than 700 homes in two days:Wildfires continued to rage Tuesday in Texas, forcing the evacuation of ...
(Texas Wildfire)	203	Wildfire destroys nearly 500 homes in Texas:Read 'Wildfire destroys nearly 500 homes in Texas' on Yahoo! News. Calmer winds ...
Topic2	470	Yahoo's CEO: 'I've Just Been Fired Over the Phone':Yahoo's cursing CEO Carol Bartz is out the door and just sent staff ...
(Yahoo's CEO)	220	Exclusive: Carol Bartz Out at Yahoo; CFO Tim Morse Named Interim CEO:According to sources at the company, Yahoo's ...
Topic3	3145	Paul Krugman Calls For Economic Sanity In A Politically Insane World: ... was on ABC today discussing President Obama's ...
(Politics)	3135	Palin in Iowa warns Tea Party ... not only against President Obama but also criticized the Republican presidential field.
Topic4	6150	Facebook Cheats, Hacks and Exploits: Restaurant City :Need help with the facebook games? Find your games cheats here. ...
(Facebook Cheats)	6113	Treasure Isle Cash and gold hack, unlimited energy and more updates to come. Facebook Myspace Cheats:Awesome Cheat Tools ...

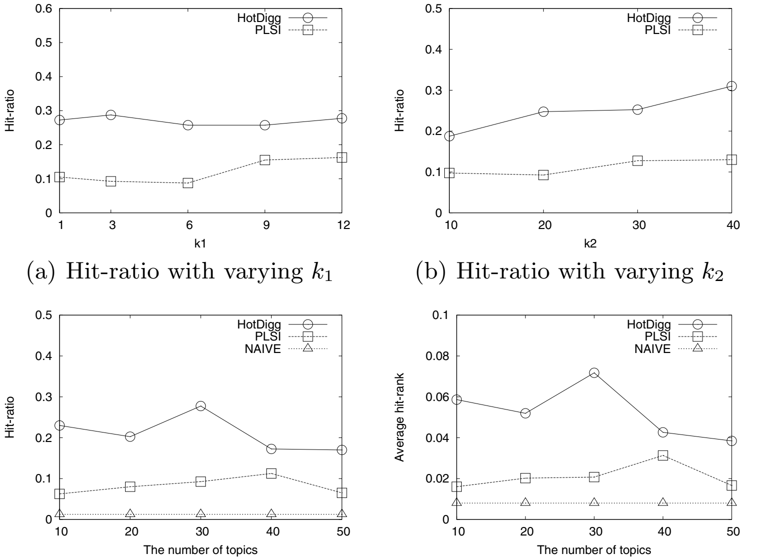
Fig. 4. The result topics and digg articles

7.4 Finding Top-k_1 Popular Topics and Top-k_2 Digg Articles

Using the test data sets, TEST-DATA(1) – TEST-DATA(4), we first conducted experiments with varying the number of topics t, the number of popular topics k_1 to find and the number of Digg articles k_2 to retrieve. Since our algorithm determines the cluster memberships of the Digg articles probabilistically, it is hard to judge clearly whether the resulting top-k_1 topics contain the popular topic that we manipulated in the test data. Thus, with the top-k_2 Digg articles found by the implemented algorithms, we evaluate the accuracy of the algorithms by computing two quality measures called *hit-rate* and *average hit-rank* [5].

With each test data set which contains the Digg articles that we amplified their scores to make a popular topic artificially, assume that we found the top-k_1 popular topics S and the top-k_2 Digg articles with our algorithms. Among the top-k_2 Digg articles found, let h be the number of articles which were selected as Digg articles related to the popular topic when we generate the test data. Let $n_T(u)$ be the number of articles whose scores are modified in the test data set, which is 20 in our test data sets. Then, the hit-rate with the top-k_2 articles retrieved is defined as $h/n_T(u)$. The average hit-rank is for measuring the effectiveness of ranking of the retrieved articles. With the retrieved top-k_2 Digg articles sorted in the decreasing order of topical scores $s_T(d_i, S)$, let p_1, p_2, ..., p_h be the position of the Digg articles which were selected for the popular topic. Then, the average hit-rank for a test user is defined as $(1/n_T(u)) \cdot \sum_{i=1}^{h}(1/p_i)$. As the Digg articles with amplified scores occur with high ranks in the top-k_2 articles, the average hit-rank gets larger.

In every graph, we plotted the average of hit-rates and average hit-rank using the result not only by running each algorithm with the 4 test data sets but also by repeating experiments 20 times for each data set. The default value of the parameters are set as $t = 20$, $k_1 = 3$ and $k_2 = 20$.



(a) Hit-ratio with varying k_1

(b) Hit-ratio with varying k_2

(c) Hit-ratio with varying t

(d) Average hit-rank with varying t

Fig. 5. Experiments with varying k_1, k_2 and t

Varying k_1: We varied the number of popular topics k_1 from 1 to 12. The hit-rates of *HotDigg* and *PLSA* algorithms are shown in Figure 5(a). As the graph illustrates, our proposed algorithm *HotDigg* outperforms *PLSI* in terms of accuracy. This is because our model utilizes additionally the property that Digg articles concerning the same topic obtain the similar digg scores to cluster the Digg articles.

The *HotDigg* algorithm shows the best performance with $k_1 = 3$, however, the performance does not change much while varying k_1. Thus, we use $k_1 = 3$ as the default value in the rest of experiments.

Varying k_2: We next varied k_2 from 10 to 40 which is the number of Digg articles to select. We plotted the hit-ratio while varying k_2 in Figure 5(b). The performances of both *HotDigg* and *PLSI* improve with increasing k_2 since, as we select more Digg articles, we simply have more chance to find the articles with amplified digg scores in the test data sets. The graph also shows that *HotDigg* performs better than *PLSI* in every value of k_2.

Varing t: With varying the number of hidden topics t from 10 to 50, we plotted the hit-rates and average hit-ranks of all three algorithms in Figure 5(c) and Figure 5(d). As the graphs illustrate, *HotDigg* shows much better performance than *PLSI* with upto 3 times (when $t = 30$). *NAIVE* shows a constant hit-ratio in all values of t since it simply selects the Digg articles with the k_2 largest digg scores without considering the hidden topics. The graph shows that the performances of both *HotDigg* and *PLSI* are much better than that of *NAIVE*.

8 Conclusion

In Digg, people share articles in web pages and vote thumbs up or thumbs down to the submitted articles. We proposed HotDigg, a system for recommending the popular and interesting topics in Digg using probabilistic modeling. Utilizing the fact that the Digg articles related to the popular topics obtains the similar voting scores, we designed a generative model that each article is produced by a topic mixture mode and its voting score is also determined by its topic. Then, we developed an estimation algorithm for learning the model parameters in our probabilistic model. By performance study, we confirmed the effectiveness of our HotDigg by comparing HotDigg with the traditional topic modeling method.

Acknowledgment. This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 2011-0000349).

References

1. Bertsekas, D.P.: Nonlinear Programming, 2nd edn., Cambridge (1999)
2. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. Journal of Machine Learning Research 3, 993–1022 (2003)
3. Delcious: Delicious (2011), http://del.icio.us
4. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. Journal of Royal Statist. Soc. 39, 1–38 (1977)
5. Deshpande, M., Karypis, G.: Item-based top-recommendation algorithms. ACM Trans. Inf. Syst. 22(1), 143–177 (2004)
6. Digg: Digg (2011), http://digg.com
7. Hofmann, T.: Probabilistic latent semantic indexing. In: SIGIR (1999)
8. Inc., D.: Digg developers (2011), http://developers.digg.com/
9. Kasiviswanathan, S.P., Melville, P., Banerjee, A., Sindhwani, V.: Emerging topic detection using dictionary learning. In: CIKM (2011)
10. Lee, D.D., Seung, H.S.: Algorithms for non-negative matrix factorization. In: NIPS (2000)
11. Lerman, K., Galstyan, A.: Analysis of social voting patterns on digg. CoRR abs/0806.1918 (2008)
12. Lerman, K., Ghosh, R.: Information contagion: An empirical study of the spread of news on digg and twitter social networks. In: ICWSM (2010)
13. Li, W., McCallum, A.: Pachinko allocation: Dag-structured mixture models of topic correlations. In: ICML, pp. 577–584 (2006)
14. Lovins, J.B.: Sourceforge: The lovins stemming algorithm (2009), http://sourceforge.net/projects/stemmers/
15. Mei, Q., Ling, X., Wondra, M., Su, H., Zhai, C.: Topic sentiment mixture: modeling facets and opinions in weblogs. In: WWW (2007)
16. Mei, Q., Zhai, C.: Discovering evolutionary theme patterns from text: an exploration of temporal text mining. In: KDD, pp. 198–207 (2005)

17. Mei, Q., Zhai, C.: A mixture model for contextual text mining. In: KDD, pp. 649–655 (2006)
18. Mitchell, T.M.: Machine Learning. WCB/McGraw-Hill (1997)
19. Morinaga, S., Yamanishi, K.: Tracking dynamics of topic trends using a finite mixture model. In: KDD (2004)
20. Wu, C.F.J.: On the convergence properties of the em algorithm. The Annals of Statistics 11(1), 95–103 (1983)

Assessing Web Article Quality by Harnessing Collective Intelligence*

Jingyu Han[1], Xueping Chen[1], Kejia Chen[1], and Dawei Jiang[2]

[1] School of Computer Science and Technology, Nanjing University of Posts and Telecommunications
Nanjing 210003, P.R.China
{hjysky,cxpsky,kejia.chen}@gmail.com
[2] School of Computing
National University of Singapore
Singapore 119077
jiangdw@comp.nus.edu.sg

Abstract. Existing approaches assess web article's quality mainly based on syntax, but seldom work is given on how to quantify its quality based on semantics. In this paper we propose a novel Semantic Quality Assessment(SQA) approach to automatically determine data quality in terms of two most important quality dimensions, namely accuracy and completeness. First, alternative context with respect to source article is built by collecting alternative web articles. Second, each alternative article is transformed and represented by semantic corpus and dimension baselines are synthetically generated from these semantic corpora. Finally, quality dimension of source article is determined by comparing its semantic corpus with dimension baseline. Our approach is promising way to assess web article quality by exploiting available collective knowledge. Experiments show that our approach performs well.

1 Introduction

Web is a tremendously vast content repository, which serves people with various formats of data. The web data quality, namely how good the web data is, is now attracting more attention. How to automatically assess web data quality is the key to make use of web data. We limit our work to web article such as Wikipedia article, online news, etc., which is a dominant format of web data.

Generally speaking, data quality is widely accepted as a multi-dimensional concept. Many efforts have been made to assess web data quality [1,2,3] but these work focuses on assessing web quality in terms of syntax rather than semantics. Furthermore, how to infer quality level using the collective knowledge of web community is also not touched on. So we propose a novel Semantic Quality Assessment (SQA) approach to automatically assess web article's quality in terms of two most important quality dimensions, namely accuracy and completeness [4,5].

* The work is supported by National Natural Science Foundation of China under grants No. 61003040, 61100135, 60970030, 60903181 and 40801149.

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 428–439, 2012.
© Springer-Verlag Berlin Heidelberg 2012

Our approach is based on the following observations. First, a web article is actually a collection of facts and this semantic corpus represents what the author expresses. Second, on the web one topic is usually described by many alternative articles. They are semantically the same or complementary to each other. Third, true facts can be derived and identified automatically from *all* the alternative articles, thus avoiding laborious manual interaction. Based on this, we propose that a source article's accuracy and completeness are gauged by the following three phases.

phase 1: Building Alternative Context. Given a source article, its alternative context is constructed with two steps. First, the relevant articles are retrieved using a set of keywords or title and they constitute a relevant space. Second, each article in relevant space is compared with the source article in terms of both topic and syntax. The articles similar to source articles constitute the alternative context. In particular, to measure the topic similarity Latent Dirichlet Allocation (LDA) analysis is exploited [15]. To measure the syntax similarity, n-gram model is used [17].

phase 2: Extracting Dimension Baselines. Each alternative article is semantically regarded as a collection of facts and each fact is represented as a tri-tuple (h, v, t), where h is the head element, t is the tail element and v is the verb connecting h and t. Based on these semantic corpora, accuracy baseline is extracted by voting and completeness baselines is synthesized by graph scoring in its alternative context.

phase 3: Computing Quality Dimensions. The quality dimensions are determined by comparing source article's semantic corpus with dimension baselines.

To sum up, the contributions of the paper are summarised as follows. First, we propose web article's quality are gauged in its alternative context, which is a collective knowledge collection relating to the topic of source article. Second, we give how to construct an article's semantic corpus and present how to synthesize accuracy baseline and completeness baseline from all the alternative articles' semantic corpora. Experiments and analysis show that our approach can correctly give web article's quality rating in terms of semantics.

2 Related Work

Data quality is an important issue to all the content contributors and its evaluation approaches can be divided into two categories. The first category focuses on *qualitatively* analysing data quality dimensions [5,4,6]. The second category deals with how to *quantitatively* assess quality of data [7,8]. The most obvious quality assurance approach is grammar check. The writer's workbench was a program to detect some quality metrics such as split infinitives, overly long sentences, wordy phrases, etc [7]. Literature [8] proposes to use Latent Semantic Analysis (LSA) algorithm to measure cohesion.

The work closely relates to ours is on assessing web article's quality. A significant number of quality indicators are exploited to assess Wikipedia article quality [1]. Basic, PeerReview and ProbReview are three schemes to quantify quality based on contributors' reputation[9]. MaxEnt [2] uses maximum entropy model to identify Wikipedia article quality. Literature [3] discusses seven IQ metrics which can be evaluated automatically on Wikipedia content. Literature [11] gives how to use revision history to assess the trustworthiness of articles. These methods focus on analysing different kinds of quality indicators and they do not touch on how to use fact semantics to identify quality level.

Another work relevant to ours is on how to extract facts or relations from web documents. Literature [12] proposes how to find out the most truthful statement relating to the given statement. Literature [13] regards the problem of concept extraction from corpora as a market-baskets problem, adapting statistical measures of support and confidence to achieve this. Literature [14] gives an iteration method to retrieve facts from web pages.

3 Building Alternative Context

An article's quality dimensions are computed in its alternative context.

Definition 1 (alternative context). *Alternative context of an article P is composed of a collection of alternative articles $\{P_1, P_2, ..., P_n\}$, each of which has a similarity $\theta < sim(P, P_i) < 1$ $(0 < \theta < 1)$ with respect to P.*

We first use the search engine to retrieve the relevant articles from the preferred web sites. The title or a set of keywords act as the query terms. The returned top K articles are regarded as the relevant ones. The alternative articles are chosen from the relevant articles. They should cover the same topic with respect to the source article. This is achieved by Latent Dirichlet Allocation (LDA) analysis [15]. Furthermore, if two articles describe the same topic, they usually have some common lexical items and they cannot syntactically differ much from each other. Hence we combine both topic and lexical features to determine alternative articles.

3.1 Training LDA Models Offline

We use the Latent Dirichlet Allocation (LDA) to model the relationship between words and articles [15]. The basic idea is that articles are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words. The whole is shown in algorithm 1. As stated in literature [16], using multiple LDA models by setting different numbers of topics outperforms single model. In our task, the following numbers of topics are used: $k = 12, 24, 48$ and 96. All the trained models are saved.

3.2 Determining Alternative Articles

The alternative articles are determined based on its topic and lexical similarity with respect to source article. That is to say, the similarity of two articles P_s and P_r is defined as follows.

$$sim(P_s, P_r) = \eta \times sim_{topi}(P_s, P_r) + (1 - \eta) \times sim_{lex}(P_s, P_r) \tag{1}$$

Algorithm 1: trainingLDA(offline)

```
Input: training articles $D = \{D_1, D_2, ..., D_M\}$, categories $\mathcal{C} = \{C_1, C_2, ..., C_n\}$
       and their relationships with categories $R \in D \times \mathcal{C}$
Output: LDA models $M = (\alpha, \beta, k)$
$M = \{\}$; // Initialize
foreach $C_j \in \mathcal{C}$ do
    $MD_j = \{w | w \in D_i, (D_i, C_j) \in R\}$;
    Processing $MD_j$ to remove stop words and rare words;
    calculate word frequency matrix $WF_j$;
end
for $k \in \{12, 24, 48, 96\}$ do
    train the LDA model $(\alpha_k, \beta_k, k)$ using $\{WF_j\}$ and $k$ topics;
    $M \leftarrow M \cup (\alpha_k, \beta_k, k)$;
end
return $M$;
```

where $0 < \eta < 1$, $sim_{topi}(P_s, P_r)$ and $sim_{lex}(P_s, P_r)$ are the topic and lexical similarity respectively.

Each article's topics distribution is given based on trained LDA models and its topic similarity with the source article is given as algorithm 2.

Algorithm 2: computeTopicSimilarity

```
Input: source document $P_s$, relevant document $P_r$, trained LDA models
       $\{(\alpha_k, \beta_k, k)\}$
Output: topic similarity $sim_{topi}(P_s, P_r)$
foreach $k \in \{12, 24, 48, 96\}$ do
    compute topic distribution $\alpha_{k,s}, \alpha_{k,r}$ based on $(\alpha_k, \beta_k, k)$;
    compute $CosSim(\alpha_{k,s}, \alpha_{k,r})$ of $P_s$ and $P_r$;
end
$sim_{topi} \leftarrow$ compute mean of top two $CosSim$ values; return $sim_{topi}$ ;
```

Given two articles, their lexical similarity is computed in n-gram multi-dimensional space [17]. Each article is regarded as a vector in n-gram space and its each component is the frequency of corresponding n-grams. Now the lexical similarity of two articles P_s, P_r is computed as

$$sim_{lex}(P_s, P_r) = CosSim(vec(P_s), vec(P_r)) \tag{2}$$

Given a source article and context threshold σ, the relevant article with sim greater than σ belongs to its alternative context.

4 Extracting Dimension Baselines in Alternative Context

Given a source article, its accuracy baseline and completeness baseline are extracted and synthesized from all its alternative articles' corpora.

4.1 Extracting Semantic Corpus for Each Alternative Article

In terms of semantics, an article represents a set of facts. Note in this paper we only discuss factual representation and how to interpret opinion representation is our future work. Without loss of generality, most of the facts are represented as a tri-tuple (h, v, t). Here h is a head element, which is a noun phrase. t is a tail element, which is a noun phrase or an adjective. v is a verb connecting head and tail elements. Extraction of facts from an article is based on parsing it using Stanford Log-Linear Part-Of-Speech tagger[1].

During extraction the facts of all alternative articles one thesaurus table is built to index the synonyms. To determine whether two nouns, pronouns or adjectives are of semantically equivalence, WordNet is consulted[2]. As for two verb patterns v_1 and v_2, their semantic equivalence is determined according to the following rules.

1. **Rule 1**: If they are totally syntactically equal, they are of equivalence. Otherwise go to rule 2.
2. **Rule 2**: By consulting WordNet, whether they are synonyms are identified. If not, go to rule 3.
3. **Rule 3**: Their similarity is computed using distributional hypothesis as algorithm 3. As stated by distributional hypothesis, if the two verbs are often distributed over a set of common word pairs, it follows that the two verbs must be semantically similar[18].

```
Algorithm 3: equalVerbPattern
Input: two verb patterns ’$v_1$’, ’$v_2$’
Output: semantic similarity of two verb patterns
$S_1 \leftarrow$ searching ”****$v_1$****” and return the top $k$ snippets;
$S_2 \leftarrow$ searching ”****$v_2$****” and return the top $k$ snippets;
$WP_1 \leftarrow$ Parsing $S_1$ and return all the word pairs enclosing $v_1$;
$WP_2 \leftarrow$ Parsing $S_2$ and return all the word pairs enclosing $v_2$;
$WP \leftarrow WP_1 \cap WP_2$;
$f_1 \leftarrow$ the frequencies of all words in WP with respect to $v_1$;
$f_2 \leftarrow$ the frequencies of all words in WP with respect to $v_2$;
return $CosSim(f_1, f_2)$;
```

Simultaneously three fact hash tables, namely Head Hash(HH), Verb Hash(VH) and Tail Hash(TH), are built to index the words in head element, verb and tail element respectively. Each hash table consists of two columns, namely *key* and *facts*. The *key* is a word in the fact element. The *facts* are a set of fact entries that contain the key.

[1] http://nlp.stanford.edu/software/tagger.shtml
[2] http://wordnet.princeton.edu/

4.2 Synthesizing Quality Dimension Baselines

```
Algorithm 4: extractCandiFacts
  Input: thesaurus table $T$, Hash tables $HH$,$VH$,$TH$, source fact $f_s$ and
         threshold $\gamma$
  Output: top $Q$ candidate facts
  $\mathit{\Gamma} \leftarrow \emptyset$; $(h, v, t) \leftarrow$ extracting each fact element from $f_s$;
  $set_1 \leftarrow$ query$(T, HH, VH, f_s, h, v, \gamma)$; $set_2 \leftarrow$ query$(T, HH, TH, f_s, h, t, \gamma)$;
  $set_3 \leftarrow$ query$(T, VH, TH, f_s, v, t, \gamma)$; $\mathit{\Gamma} \leftarrow set_1 \cup set_2 \cup set_3$;
  Sort the facts in $\mathit{\Gamma}$ based on similarity with respect to $f_s$;
  return top $Q$ ones of $\mathit{\Gamma}$;
```

Constructing Accuracy Baseline. Source article's accuracy baseline consists of a collection of most accurate facts, each of which corresponds to one distinct fact in source article. It is extracted by searching the thesaurus table and three fact hash tables. It contains two phases, namely *producing candidate facts* and *identifying target*. In producing candidate facts, all the candidate facts similar or complementary to source fact are extracted based on the fact similarity.

Definition 2 (fact similarity). *Given two facts* $\overline{f}(\overline{h}, \overline{v}, \overline{t})$, $\underline{f}(\underline{h}, \underline{v}, \underline{t})$, *their fact similarity is*

$$sim_{fact}(\overline{f}, \underline{f}) = \frac{1}{3}pss(\overline{h}, \underline{h}) + \frac{1}{3}pss(\overline{v}, \underline{v}) + \frac{1}{3}pss(\overline{t}, \underline{t}) \tag{3}$$

where pss is the phrase semantic similarity defined as follows.

Definition 3 (phrase semantic similarity). *Given two noun phrases, verb phrases or adjective phrases* $ph_1 = w_{a1}w_{a2}, ..., w_{an}$, $ph_2 = w_{b1}, w_{b2}, .., w_{bm}$, *their phrase semantic similarity is defined as*

$$pss(ph_1, ph_2) = \frac{|ph_1 \cap ph_2|}{|ph_1 \cup ph_2|} \tag{4}$$

where $|ph_1 \cap ph_2|$ *is the number of words which are semantically equivalent and* $|ph_1 \cup ph_2|$ *is the total number of words they contain.*

The candidate facts are identified by searching hash tables. Each time two components of source fact act as query terms. Then, all the found facts are sorted and the top Q ones are returned. The whole procedure is described in algorithm 4. Here the routine *query* is to retrieve the candidate facts based on two components of source fact, which is described in algorithm 5.

In identifying target phase the most accurate fact is identified by voting. The confidence is used to measure how the components of a fact are confirmed by other candidate facts. It is a combination of head, verb and tail confidence.

$$conf_h(f(h, v, t)) = \frac{|h \cup v \cup t|}{|v \cup t|} \tag{5}$$

Algorithm 5: query

```
Input: thesaurus table $T$, two hash tables, source fact $f_s$ and its two
       components $e_1$ and $e_2$, similarity threshold $\gamma$
Output: candidate facts
$F \leftarrow \emptyset$; $S_1 \leftarrow$ retrieve $e_1$' synonyms from $T$;$S_2 \leftarrow$ retrieve $e_2$' synonyms from $T$;
foreach $s_1 \in S_1$ do
    $F_1 \leftarrow$ retrieve facts from one of $\{HH, VH, TH\}$;
    foreach $s_2 \in S_2$ do
        $F_2 \leftarrow$ retrieve facts from one of $\{HH, VH, TH\}$; $F \leftarrow F \cup (F_1 \cap F_2)$;
    end
end
foreach $f \in F$ do
    if $sim_{fact}(f_s, f) < \gamma$ then
        remove $f$ from $F$;
    end
end
return $F$;
```

$$conf_v(f(h,v,t)) = \frac{|h \cup v \cup t|}{|h \cup t|} \tag{6}$$

$$conf_t(f(h,v,t)) = \frac{|h \cup v \cup t|}{|h \cup v|} \tag{7}$$

Here $|h \cup v \cup t|$ is the number of facts that are semantically equal to fact $f(h,v,t)$. $|v \cup t|$, $|h \cup t|$ and $|h \cup v|$ denote the number of facts that are only semantically equal to sub-components (v,t), (h,t) and (h,v) respectively. Now the confidence of a fact f is

$$conf(f) = \omega_1 conf_h(f) + \omega_2 conf_v(f) + \omega_3 conf_t(f) \tag{8}$$

where $\omega_1 + \omega_2 + \omega_3 = 1$.

Based on the confidence, the most accurate fact is determined with algorithm 6.The accuracy baseline is constructed by the algorithm 7.

Constructing Completeness Baseline. The completeness baseline of a source article is constructed as a graph, each vertex of which corresponds to one distinct fact in alternative context. It is constructed with two steps.

step 1: Each fact is inserted into completeness graph, acting as one vertex with an initial completeness score $s(v_i, 0) = 1$. Simultaneously weighted edges denoting fact similarities are added to connect the semantically similar facts.
step 2: Iterate computing the graph vertex scores with the equation 9.

$$s(v_i, t+1) = s(v_i, t) - \frac{1}{2^i} \sum_{j \in con(v_i)} \frac{s(v_j, t)}{|con(v_i)| + \sum_{v_k \in con(v_j)} \frac{1}{sim_{fact}(v_k, v_j)}} \tag{9}$$

Algorithm 6: identifyTarget

```
Input: candidate facts $\Gamma$, source fact $f_s$, confidence threshold $\theta$
Output: most accurate fact
if $conf(f) < \theta(\forall f \in \Gamma \setminus f_s)$ then
    return $null$;
end
else
    $conflist \leftarrow$ sorts all facts $\in \Gamma$ based on its confidence;
    $f_{top} \leftarrow$ top fact of $conflist$;
    if $conf(f_{top}) > \theta$ then
        return $f_{top}$;
    end
    else
        return $f_s$;
    end
end
```

Algorithm 7: constructAccuracyBaseline

```
Input: corpus of source article $corp(P)$, thesaurus $T$, hash tables $HH$, $VH$,
       $TH$, threshold $\gamma$ and $\theta$
Output: accuracy baseline
$\Pi \leftarrow \emptyset$; // Initialize the accuracy baseline
foreach $f_s \in corp(P)$ do
    $\Gamma \leftarrow extractCandiFacts(T, HH, VH, TH, f_s, \gamma)$;
    $f_{acc} \leftarrow identifyTarget(\Gamma, f_s, \theta)$; $\Pi \leftarrow \Pi \cup f_{acc}$;
end
return $\Pi$;
```

where $s(v_i, t)$ is the score of vertex v_i at the t-th iteration, $sim_{fact}(v_k, v_j)$ is the fact similarity between v_k and v_j and $con(v_i)$ is the vertices that are directly connected to vertex v_i. Given a convergence threshold ρ, if the average score variance of all vertices between two successive iterations are within ρ, the computation ends.

Actually the graph vertex score denotes the information coverage of the fact. If there is an edge between two vertices, it means the information one vertex conveyed is also somewhat implied by the other node. During the iteration, the score of each vertex becomes smaller and smaller. When the iteration ends, the sum is the amount of information the whole alternative context holds.

5 Computing Quality Dimensions

5.1 Computing Accuracy

Accuracy gives to what extent the data is close to corresponding true facts.

Definition 4 (accuracy). *Given an article P with n facts, its accuracy is defined as*

$$acc(P) = \frac{1}{n}\sum_{s=1}^{n} sim_{fact}(f_s, f_b) \tag{10}$$

where f_b is f_s's corresponding fact in accuracy baseline.

5.2 Computing Completeness

Completeness gives to what extent the related facts are described in the source article. It is measured by comparing itself with its completeness baseline.

Definition 5 (completeness). *Given an article P, its completeness is*

$$comp(P) = \frac{|P|}{|B|} \tag{11}$$

where $|P|$ and $|B|$ are the amount of information source article and the completeness baseline contains respectively.

Two policies are proposed to quantify completeness as follows.

Plain Policy to Quantify Completeness(PC). We assume that each unique fact in alternative context plays an equal role in the coverage of information. Suppose that source article contains n non-duplicate facts and the baseline has N non-duplicate facts, then we have

$$comp(P) = \frac{n}{N} \tag{12}$$

Weighted Policy to Quantify Completeness(WC). Different facts play different roles in the coverage of information. Given a source graph P and its corresponding baseline B, the completeness is quantified as

$$comp(P) = \frac{\sum_{v_i \in P} s(v_i)}{\sum_{v_j \in B} s(v_j)} \tag{13}$$

where $s(v_i)$ and $s(v_j)$ are the final completeness scores of source article vertex and baseline vertex respectively.

6 Experiment Results

We first downloaded 782 source articles on scientists from Wikipedia. The articles have been assigned quality class labels, namely Featured Article(FA), Good Article(GA), B-Class(B), C-Class(C), Start-Class(ST) and Stub-Class(SU) , according to Wikipedia community quality grading scheme[3]. For each source article, we typed its title into the search box of google. The top 10 returned were regarded as the relevant ones. So there are totally 8602 web articles in the dataset. The data are first transformed into plain text by removing HTML tags

[3] en.wikipedia.org/wiki/Wikipedia:Version_1.0_Editorial_Team/Assessment

and figures. Then, they are fed into building alternative context module. We set $\eta = 0.5$, that is to say both topic and syntax weigh the same. As for the syntax similarity, 3-gram space is employed.

6.1 Precision of Quality Ranking

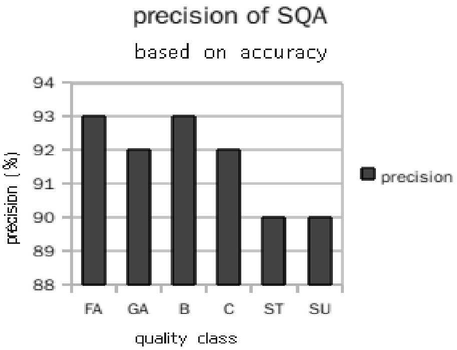


Fig. 1. Precision Based on Accuracy

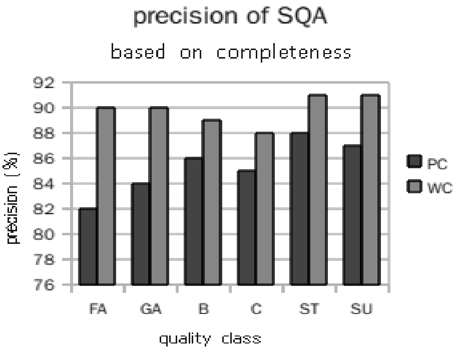


Fig. 2. Precision Based on Completeness

The dataset is composed of N articles and $\sum_{i=1}^{6} N_i = N$ holds, where N_1, $N_2, ..., N_6$ denote the numbers of articles in FA, GA, B, C, ST, SU classes respectively. We suppose the linear order based on the dimension scores is $< P_1, P_2, ..., P_N >$. It naturally means the first N_1 articles belong to FA class, the subsequent N_2 articles belong to GA class, and so on. Thus the precision with respect to each quality class i is

$$prec(i) = \frac{|X \cap Y|}{|Y|} \tag{14}$$

where Y is the articles actually in quality class i and X is the articles that are classified into i class by SQA.

Figure 1 gives the precision per quality class based on accuracy and Figure 2 gives that based on completeness. We see that the performance based on accuracy is better than that based on completeness. Based on the accuracy dimension the percentage of correct ratings ranges between 90% and 93% and it performs better on FA, GA, B, C classes than on ST, SU classes. Based on the completeness dimension, the SQA also gives an acceptable precision values which are between 82% and 91% . We can see that WC policy consistently outperforms PC policy by a percentage between 3% and 8%. Figure 3 gives the precision per quality class based on combination of accuracy and completeness. Note here the weights for accuracy and completeness are tuned as 0.62 and 0.38 respectively. Obviously SQA approach gives the best performance when accuracy and completeness are combined. Specifically, the average precision ranges between 93% and 98%. In particular, the precision given by acc.+WC is always above 94%.

6.2 Comparison with Previous Work

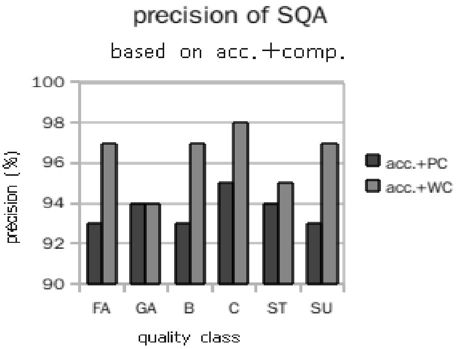


Fig. 3. Precision Based on Acc.+Compl.

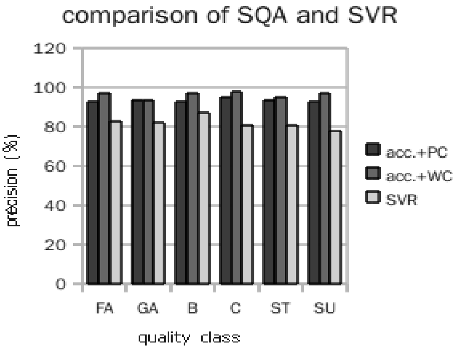


Fig. 4. Performance Comparison

To the best of our knowledge, most work assesses web article quality based on non-semantic characteristics. We compare our SQA approach with the state-of-the-work, namely SVR approach [1]. We implemented it ourselves. To be fair, we implemented SVR approach using the group of features that was reported to perform best, namely structure feature group. Figure 4 reports the comparison between the SVR and our SQA. We can see the SQA performs fairly better than the SVR. Specifically acc.+PC policy outperforms the SVR approach by a percentage between 6 and 15 and acc.+WC policy outperforms the SVR approach by a percentage between 12 and 19.

7 Conclusion and Future Work

To make use of vast amount of web data, how to automatically give the quality level is a pressing concern. We propose to assess web article's quality in terms of fact semantics by collecting collective knowledge. It is a promising method for assessing web article quality. Its advantage is as follows. First, our SQA approach is an automatic web quality rating solution, which make full of related web knowledge provided by web users. Second, it provides a viable way to assess web article quality in terms of fact semantics, which gives a more precise rating of data quality. Experiments show that our approach gives a satisfying performance.

In future we will tackle how to process opinion representation, how to more efficiently synthesize the baseline and how to extract facts more precisely.

References

1. Dalip, D.H., Cristo, M., Calado, P.: Automatic quality assessment of content created collaboratively by web communities: A case study of wikipedia. In: Proc. of JCDL 2009, pp. 295–304 (2009)
2. Rassbach, L., Pincock, T., Mingus, B.: Exploring the feasibility of automatically rating online article quality (2008)
3. Stvilia, B., Twidle, M.B., Smith, L.C.: Assessing information quality of a community-based encyclopedia. In: Proc. of the International Conference on Information Quality, pp. 442–454 (2005)
4. Wang, R.Y., Kon, H.B., Madnick, S.E.: Data quality requirements analysis and modelling. In: Proc. of the 9th ICDE, pp. 670–677 (1993)
5. Aebi, D., Perrochon, L.: Towards improving data quality. In: Proc. of the International Conference on Information Systems and Management of Data, pp. 273–281 (1993)
6. Bouzeghoub, M., Peralta, V.: A framework for analysis of data freshness. In: Proc. of IQIS 2004, pp. 59–67 (2004)
7. Macdonald, N., Frase, L., Gingrich, P., Keenan, S.: The writer's workbench: computer aids for text analysis. IEEE Transactions on Communications 30(1), 105–110 (1982)
8. Foltz, P.W.: Supporting content-based feedback in on-line writing evaluation with lsa. Interactive Learning Environments 8(2), 111–127 (2000)
9. Hu, M., Lim, E.P., Sun, A., Lauw, H.W., Vuong, B.Q.: Measuring article quality in wikipedia: Models and evaluation. In: Proc. of the Sixteenth ACM Conference on Information and Knowledge Management, pp. 243–252 (2007)
10. Zeng, H., Alhossaini, M.A., Ding, L.: Computing trust from revision history. In: Proc. of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services (2006)
11. Zeng, H., Alhossaini, M.A., Fikes, R.: Mining revision history to assess trustworthiness of article fragments. In: Proc. of International conference on Collaborative Computing: Networking, Applications and Worksharing, pp. 1–10 (2009)
12. Li, X., Meng, W., Yu, C.: T-verifier: Verifying truthfulness of fact statements. In: Proc. of ICDE 2011, pp. 63–74 (2011)
13. Parameswaran, A., Rajaraman, A., Garcia-Molina, H.: Towards the web of concepts: Extracting concepts from large datasets. In: Proc. of 2010 VLDB, vol. 3, pp. 566–577 (2010)
14. Zhao, S., Betz, J.: Corroborate and learn facts from the web. In: Proc. of KDD 2007, pp. 995–1003 (2007)
15. Blei, D., Ng, A., Jordan, M.: Latent dirichlet allocation. The Journal of Machine Learning Research 3, 993–1022 (2003)
16. Hofmann, T.: Probabilistic latent semantic analysis. In: Proc. of UAI 1999 (1999)
17. Kukich, K.: Technique for automatically correcting words in text. ACM Computing Surveys 24(4), 377–439 (1992)
18. Harris, Z.: Distributional structure. Word 10(23), 146–162 (1954)

Context Sensitive Tag Expansion with Information Inference

Hongyun Cai, Zi Huang, Jie Shao, and Xue Li

School of Information Technology and Electrical Engineering
The University of Queensland, Australia
caihongyun@gmail.com, {huang,jshao,xueli}@itee.uq.edu.au

Abstract. The exponential explosion of web image data on the Internet has been witnessed over the last few years. The precise labeling of these images is crucial to effective image retrieval. However, most existing image tagging methods discover the correlations from tag co-occurrence relationship, which leads to the limited scope of extended tags. In this paper, we study how to build a new information inference model over image tag datasets for more effective and complete tag expansion. Specifically, the proposed approach uses modified Hyperspace Analogue to Language (HAL) model instead of association rules or latent dirichlet allocations to mine the correlations between image tags. It takes advantage of context sensitive information inference to overcome the limitation caused by the tag co-occurrence based methods. The strength of this approach lies in its ability to generate additional tags that are relevant to a target image but may have weak co-occurrence relationship with the existing tags in the target image. We demonstrate the effectiveness of this proposal with extensive experiments on a large Flickr image dataset."

1 Introduction

With the rapid growth of popularity in various kinds of tagging systems [10,13], tags occupy an important position in many different areas (e.g., [1,2,3]). For instance, image tagging is the crux of text-based image navigation and searching systems [8] and the performance of the image retrieval engine is largely related to the accuracy of image tags. However, manually tagging images is time-consuming. A study on Flickr [3] has shown that, although generally users are willing to provide tags, most images are only with 1∼3 tags from user annotations [15]. Thus, automatic image tag expansion has become a fundamental research topic recently.

Currently, most of semantic image tag expansion methods are applying association rules [11] or other techniques such as latent dirichlet allocation (LDA) [12] to mine image tags. Deriving one tag from another one mainly relies on co-occurrence of these two individual tags, which limits the candidates of expanded tags to small scopes (in the tag co-occurrence lists with high frequencies) and thus some hidden strong relationships between two tags with low co-occurrence frequency will be lost.

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 440–454, 2012.
© Springer-Verlag Berlin Heidelberg 2012

To overcome this problem, in this paper we present a context sensitive tag expansion method by building an information inference model. Our proposed method can be divided into two main components: an offline tags mining process and an online tag expension process. In the offline process, unweighted Hyperspace Analogue to Language (HAL) models are constructed on the image tags. In the online processing, given an image with few tags (seed tags), the concept combination of all these seed tags is computed in the form of high-dimensional HAL vectors. Then the relevance between the combined concept and each word in the dataset vocabulary is calculated based on three factors, including co-occurrence relationship, inverse vector frequency and contextual similarity. Finally, the top K most relevant words are picked up as the expended tags for the target image. The most important characteristic of this approach is its ability to generate additional tags that are relevant to a target image but may have very low co-occurrence frequencies with seed tags in the target image. The proposed approach uses modified HAL model to infer image tags instead of using association rules or LDA, which takes advantage of information inference for image tag expansion to overcome the limitation caused by the tag co-occurrence based similarity metrics. The approach would be beneficial for image sharing websites including Flickr where it can be used to assist users to provide tags at upload time by giving them a list of possible candidates to choose from.

The remainder of this paper is organized as follows. Related work on tag expansion and information inference are reviewed in Section 2, followed by the description of HAL model, and information flow in Section 3. Our improved degree algorithm of information flow model and the proposed tag expansion approach are presented in Section 4. In Section 5 we present our evaluation results along with comparison to three proposed models and two other state-of-the-art algorithms in tag expansion and information inference. Finally, we summarize the work in this paper in Section 6.

2 Related Work

The main objective of our work is to improve the process of image tag expansion via exploiting information inference. In this section, we review research work related to image tag expansion and information inference.

2.1 Image Tag Expansion

Image tag expansion is a process to automatically expand extra descriptive tags according to one or more existing tags of an image in accordance with the expanding rules generated by antecedent tags mining process. These expanded tags could be used as a personalized tag recommendation to the users of a tagging system [13,10], or a collective tag recommendation which acts as the results of a query expansion to improve the efficiency of image retrieval relied on tags [8].

Tags mining could be processed based on semantic similarity [15,20], visual similarity [14,21] or both [18,19]. We focus our review on the first one on account of the fact that our proposal does not include analysis of low-level image visual features. Sigurbjörnsson and Zwol [15] formed and analyzed a tag characterization of Flickr users' tagging behaviors, and presented four different tag recommendation strategies by means of using global asymmetric tag co-occurrence metrics to mine tags relationships. Two tag aggregation strategies (voting and summing) along with a promotion function were proposed to construct an efficient tag recommendation system without introducing tag-class specific heuristics. In [20], Xu et al. also obtained semantic similarity between tags based on tag co-occurrence. To improve the performance of tag recommendation, an authority score based on the history tagging behavior was assigned to each user and co-occurring tags which have been assigned by one user were rewarded according to the authority score while those assigned by different users were penalized. Heymann et al. [11] introduced an method of mining association rules in market-basket model for tag recommendation. In [12], rather than association rules, latent dirichlet allocation was investigated for collective tag recommendation. Latent topics were elicited from the collaborative tagging efforts of multiple users, thus solving the cold start problem for tagging new resources with only a few tags. However, the semantic similarity metrics of these methods mainly relied on tag co-occurrence, thus they all suffer from one problem: the tightly related tags may not necessarily have strong co-occurrence relationship. This is the situation especially when two tags have semantic relationships but have never occurred together. Such kind of information between tags is lost via using traditional tags mining process.

Our work mainly focuses on employing concepts from information inference to process tags mining to overcome the limitations of association rule based tagging algorithms and also discover the hidden tags relationship to improve the accuracy of recommended tags.

2.2 Information Inference

Inferential information content was first formalized by Barwise and Seligman [5] in 1997, followed by a cognitive model which was combined of three levels including symbolic processing [9]. Based on these two definitions, Song and Bruza [16] focused on the information inferences which "can be drawn on the basis of words seen in the context of other words under the proviso that such inferences correlate with corresponding human information inference" and thus proposed *context sensitive information inference*. A representational model of semantic memory named Hyperspace Analogue to Language (HAL) was introduced, along with a specifically-designed corresponding heuristic concept combination. Furthermore, an HAL-based information flow was defined and evaluated by the effectiveness of query models.

This HAL construction model is designed for analyze document corpus, thus it is not very suitable for image tags which only contain limited tags in an image. To build HAL model on image tags, we modified the HAL construction algorithm in this work. The technical details will be discussed in the next section.

3 Preliminaries

3.1 Conceptual Space Construction

Hyperspace Analogue to Language (HAL) model [7,6] is a computational model to represent a conceptual space by high-dimensional vectors based on a given corpus. Similar to the Vector Space Model (VSM), each dimension in the conceptual space corresponds to a word in the given vocabulary with N words. In HAL model, the whole corpus is scanned by a fix-sized sliding window, based on which a co-occurrence matrix is created to represent the relationships between the words within the window. The strength of the relationship between two words is inversely proportional to the distance between them. After scanning the entire corpus, an accumulated co-occurrence matrix for all words is created. Since the words in a sliding window can be considered as a certain context, the HAL model represents an accumulation of experience of the contexts in which the words appear. Given a word c, it is eventually represented as a normalized weighted vector $c =< w_{c,1}, ..., w_{c,N} >$, where $w_{c,i}$, $i = 1..N$ is the strength of association between the word c and the i-th word in the vocabulary computed from the global co-occurrence matrix. The higher $w_{c,i}$, the more frequently the word c appears with the i-th word in the same context (i.e., same sliding window).

3.2 Concept Combination

Words in different contexts may carry different concepts. For example, given the word "penguin", if the surrounding words are "ocean","Antarctica", and etc., "penguin" has strong association with "creature" or "bird". On the other hand, if the surrounding words are "book", "publishing" and etc., most likely "penguin" indicates the publisher "Penguin Books". Given a group of words, they are the triggers of each other to reveal new concepts. A concept can be a real word, like *penguin*, or a virtual word defined by a group of concepts. Concept combination is basically a vector operation, which was first introduced in [16].

Definition 1 (Concept Combination). *Given two concepts $c_1 =< w_{c_1,1}, ...,$ $w_{c_1,N} >$ and $c_2 =< w_{c_2,1}, ..., w_{c_2,N} >$, the new concept derived by combining c_1 and c_2 is denoted as $c_1 \oplus c_2 =< w_{c_1 \oplus c_2,1}, ..., w_{c_1 \oplus c_2,N} >$. The weight of the k^{th} dimension in the new concept, $w_{c1 \oplus c_2,k}$ is computed by the follows:*

$$w_{c_1 \oplus c_2,k} = (l_1 + \frac{l_1 \times w_{c_1,k}}{\max\limits_{i=1..N} w_{c_1,i}} + l_2 + \frac{l_2 \times w_{c_2,k}}{\max\limits_{i=1..N} w_{c_2,i}}) \times \alpha \tag{1}$$
$$l_1, l_2 \in (0.0, 1.0] \text{ and } l_1 > l_2$$
$$\alpha \geq 1.0$$

where α is a multiplier to emphasize the weights of dimensions appearing in both c_1 and c_2. If both $w_{c_1,k}$ and $w_{c_2,k}$ are greater than a threshold θ, α is set as a value > 1.0, otherwise, $\alpha = 1$. The symbol "$\oplus$" stands for the operation of combination.

The parameters used in concept combination are set as $l_1 = 0.5$, $l_2 = 0.3$, $\alpha = 2$, and $\theta = 0$ in [16] to achieve the best performance.

Concept combination on a group of concepts can be derived by recursively applying the combination operation on two concepts, such as $c_1 \oplus c_2 \oplus c_3 = (c_1 \oplus c_2) \oplus c_3$.

3.3 Information Inference

Given a group of words $\{c_1, ..., c_m\}$, the resulting combined concept is denoted as $\oplus c_i$, which is usually just a virtual word represented by a weighted vector. How to find the real words from the vocabulary strongly associated with the concept $\oplus c_i$ is important. An HAL based information flow is used in [16] for information inference.

Definition 2 (HAL-Based Information Flow). *Given a combined concept $\oplus c_i$, it implies c_j, iff degree($\oplus c_i \lhd c_j$)$\geq \lambda$, where*

$$degree(\oplus c_i \lhd c_j) = \frac{\sum_{k \in Q(\oplus c_i) \cap Q(c_j)} w_{\oplus c_i, k}}{\sum_{k \in Q(\oplus c_i)} w_{\oplus c_i, k}} \tag{2}$$

$Q(c)$ denotes the set of dimensions of c with the weight greater than θ and λ is a threshold value predefined by user.

$degree(\oplus c_i \lhd c_j)$ reflects the ratio of intersecting dimensions of $\oplus c_i$ and c_j. The underlying idea of Definition 2 is that if the majority of the most important properties of $\oplus c_i$ appear in c_j, c_j must have strong association with $c_1, ..., c_m$. By calculating and ranking the *degree*s between a combined concept and the real words from the vocabulary, we can return a list of words at the topmost rank as a recommendation. For example, given the words “penguin”, “book”, and “publishing”, “publisher” is the most relevant word with the largest *degree* with penguin⊕book⊕publishing. We can also have another two examples, such as river⊕clean-up⇒<flood, garbage, recover> and river⊕sunshine⇒<holiday, sky>.

4 Context Sensitive Tag Expansion

Tagging has been playing an important role in web information retrieval by annotating various web sources. Taking image tagging as an example, to facilitate an efficient semantic based image search, it is crucial to assign meaningful descriptors (tags) to images. However, manual image tagging is extremely labor consuming yet the performance of content-based automatic image tagging is unsatisfactory due to the problem of “semantic gap”. Thus efficiently recommending accurate and meaningful tags to users is critical. Here, we propose a context sensitive model for tag expansion. Given a few seed tags, a group of new tags will be expanded by analyzing the underlying concepts of the seed tags and discovering the information inference. A list of notations used in this paper is shown in Table 1.

Table 1. Notations

Notation	Description
c, c_i, c_j	individual concept
$\oplus$	concept combination operator
$\oplus c_i$	a combined concept
N	the size of vocabulary
$< w_{c,1}, ..., w_{c,N} >$	weighted vector of concept c

4.1 Image Conceptual Space

The general idea of HAL is to represent the concept of one word by the statistics of its appearance within the context of other concepts. Traditionally, HAL is used in text information retrieval. The HAL space is built on documents. In this paper, we aim to construct a conceptual space on images based on their tags. Usually, Flickr images are associated with a set of user-defined tags. We consider each image as a document, where each tag of the image is a word. In HAL model, a fix-sized sliding window is applied to scan the whole corpus. The words within the sliding window are assumed to have stronger relationships with each other than with the words outside the window. However, given an image, all its tags are supposed to describe its characteristics or semantics. It means that all the tags of an image have strong relationships with each other to some extent. Thus, a flexible-sized sliding window is applied, which scans the corpus image by image. When scanning an image, the size of the sliding window is exactly the number of the associated tags. Since the number of tags is different from image to image, the size of the sliding window is flexible. After scanning the entire image set, the global co-occurrence matrix of tags is constructed by accumulating the local co-occurrence matrix built on each sliding window. Based on the global co-occurrence matrix, the weighted vectors of tags can be generated. The concept combination of two or more tags is calculated according to Equation (1) and implemented in Algorithm 1. Example 1 illustrates the details step by step.

Example 1 (Image Conceptual Space). Given three images I_1, I_2 and I_3 associated with tags <river, fish, sunshine, kids, birds>, <river, fish, holiday> and <river, fish, birds> respectively, the global co-occurrence matrix built on I_1, I_2 and I_3 is shown in Table 2. The dimensions of the image conceptual space constructed on I_1, I_2 and I_3 are "river", "fish", "sunshine", "kids", "birds", and "holiday". The weighted concept vector of "river" is

river=<fish:0.75, sunshine:0.25, kids:0.25, birds:0.50, holiday:0.25>

Take the dimension "fish" as an example. The weighted value on dimension "fish" of "river" is calculated as $3/\sqrt{3^2 + 1^2 + 1^2 + 2^2 + 1^2} = 0.75$.

4.2 Tag Expansion with Information Inference

Given an image associated with a small number of seed tags, we propose to expand them to a group of meaningful tags by discovering the information

Algorithm 1. Concept Combination

```
Input:
    $c_1 < w_{c_1,1}, w_{c_1,1}, \ldots, w_{c_1,n} >$ (concept vector),
    $c_2 < w_{c_2,1}, w_{c_2,1}, \ldots, w_{c_2,n} >$ (concept vector),
    $l_1$ (rescale parameter for concept $c_1$),
    $l_2$ (rescale parameter for concept $c_2$),
    $\theta$ (threshold value for quality dimensions of a concept),
    $\alpha$ (emphasize multiplier).
Output:
    $c$ (combined concept of $c_1$ and $c_2$).
Description:
for each weight $w_{c_1,i}$ in $c_1$ do
    $w_{c_1,i} = l_1 + (l_1 \times w_{c_1,i})/max_k(w_{c_1,k})$;
end for
for each weight $w_{c_2,i}$ in $c_2$ do
    $w_{c_2,i} = l_2 + (l_2 \times w_{c_2,i})/max_k(w_{c_2,k})$;
end for
for each dimension $i$ in $c_1$ do
    if $w_{c_1,i} > \theta$ and $w_{c_2,i} > \theta$ then
        $w_{c_1,i} = \alpha \times w_{c_1,i}$, $w_{c_2,i} = \alpha \times w_{c_2,i}$;
    end if
end for
for each dimension $i$ in $c$ do
    $w_{c,i} = w_{c_1,i} + w_{c_2,i}$;
end for
for each dimension $i$ in $c$ do
    $w_{c,i} = w_{c,i}/\sqrt{\sum_1^n (w_{c,k})^2}$;
end for
return c
```

inference between them. Each tag stands for a concept, which can be represented by a weighted vector in the image conceptual space. Three inference models are proposed for tag expansion via information flow and various weighting schemes.

The highly weighted dimensions in the concept vector of a tag c are the tags frequently co-occurring with c in the corpus. The larger the weight, the stronger the association. However, some generic tags frequently appear in the whole collection, resulting in a high weight to most of tags. Motivated by the concept of Inverse Document Frequency (IDF) in text mining, [17] proposed the Inverse Vector Frequency (IVF) to measure the information carried by a word, which is formally defined as:

$$IVF(c) = \frac{log(\frac{N+0.5}{n})}{log(N+1)} \tag{3}$$

where N is the total number of tags in the vocabulary and n is the number of tags co-occurring with c in the same image. The higher the occurrence frequency, the less the information c carries. By considering both the specific contribution to a concept and the general information it carries, a TF/IVF inference model is designed.

Table 2. Co-occurrence Matrix

	river	fish	sunshine	kids	birds	holiday
river		3	1	1	2	1
fish	3		1	1	2	1
sunshine	1	1		1	1	
kids	1	1	1		1	
birds	2	2	1	1		
holiday	1	1				

Definition 3 (TF/IVF Inference Model). *Given a combined concept $\oplus c_i$ and the j-th tag c_j in the vocabulary, $\oplus c_i$ implies c_j, iff*

$$w_{\oplus c_i,j} \times IVF(c_j) > \lambda \tag{4}$$

where $w_{\oplus c_i,j}$ stands for the contribution (i.e., the strength of the association) to $\oplus c_i$ from c_j and λ is a threshold value. If $w_{\oplus c_i,j} = 0$, $w_{\oplus c_i,j}$ is set as the minimum positive weight of $\oplus c_i$.

This model only considers the co-occurrence of $\oplus c_i$ and c_j, yet the context information is not taken into account. In some cases, $w_{\oplus c_i,j}$ is relatively small, which means c_j rarely appears with $\oplus c_i$. According to Definition 3, most likely c_j will not be derived from $\oplus c_i$. However, if the context of c_j's appearances is similar to the context of $\oplus c_i$, an implicit correlation is expected to exist between them, which has been evidenced in [17,7]. The IVF-HAL model [17] is proposed to capture the context information and discover the implicit relationship between words appearing in similar contexts.

Definition 4 (IVF-HAL Inference Model). *Given a combined concept $\oplus c_i$, it implies c_j, iff*

$$IVF(c_j) \times degree(\oplus c_i \lhd c_j) \geq \lambda \tag{5}$$

where $degree(\oplus c_i \lhd c_j)$ is defined in Equation 2 and λ is a threshold value predefined by user.

IVF-HAL inference model is an upgraded version of HAL-based inference model introduced in Definition 2. The original weighting scheme of HAL is frequency biased. The high frequency words always obtain high weights in any concept vectors, even though they may not be much informative. By introducing IVF, the effect of high frequent words is decreased. The experiment results also confirm that IVF-HAL inference model is superior than the original HAL model.

However, IVF-HAL model considers the context information only, while the absolute contribution of c_j to $\oplus c_i$ is neglected. To discover both the explicit and implicit relationship between concepts for an accurate information inference, a novel inference model is proposed to take into account not only the information carried by c_j via TF/IVF weighting scheme, but also the closeness of the context in which c_j and $\oplus c_i$ appear.

Definition 5 (TF/IVF Context Sensitive Inference Model). *Given a combined concept $\oplus c_i$, it implies c_j, iff*

$$w_{\oplus c_i,j} \times IVF(c_j) \times degree(\oplus c_i \lhd c_j) \geq \lambda \tag{6}$$

where $degree(\oplus c_i \lhd c_j)$ is defined in Equation 2 and λ is a threshold value predefined by user. If $w_{\oplus c_i,j} = 0$, $w_{\oplus c_i,j}$ is set as the minimum positive weight of $\oplus c_i$.

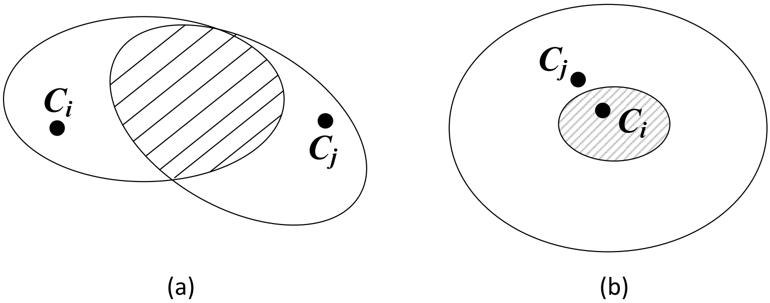


Fig. 1. An explanation of context relationship between two concepts

This model leverages the advantages from both TF/IVF weighting scheme and the context sensitive information flow. When we conduct information inference on $\oplus c_i$ and c_j, there are four possible situations.

1. The first case is that $\oplus c_i$ and c_j rarely co-occur and the contexts of their appearances are different. In this case, both $w_{\oplus c_i,j}$ and $degree(\oplus c_i \lhd c_j)$ are very low, resulting in that c_j will not be inferred by $\oplus c_i$.
2. The second case is that $\oplus c_i$ and c_j are highly associated, resulting in high $w_{\oplus c_i,j}$ and $degree(\oplus c_i \lhd c_j)$. Thus c_j will be derived from $\oplus c_i$.
3. The third case is that $\oplus c_i$ and c_j rarely co-occur but the contexts of their appearances are similar (dashed area in Figure 1(a)), resulting in a low $w_{\oplus c_i,j}$ but a high $degree(\oplus c_i \lhd c_j)$. If we apply TD/IVF model, c_j cannot be derived due to the low $w_{\oplus c_i,j}$. By involving context similarity in the proposed model, a high $degree(\oplus c_i \lhd c_j)$ compensates the information loss caused by the low co-occurrence.
4. The last case is that $\oplus c_i$ frequently co-occurs with c_j, however $degree(\oplus c_i \lhd c_j)$ is very low (Figure 1(b)). It means that $\oplus c_i$ is a quite narrow concept contained by c_j. For example, "cat" (i.e., $\oplus c_i$) appears frequently with "animal" (c_j), while the context of "animal" appearing is much broader than it of "cat" (shadow area in Figure 1(b)). If applying IVF-HAL model, the information inference between "cat" (i.e., $\oplus c_i$) and "animal" (c_j) cannot be proceeded, because IVF-HAL model overemphasizes the effect of $degree(\oplus c_i \lhd c_j)$, which is supposed to be balanced by $w_{\oplus c_i,j}$ in our proposed model.

A comprehensive performance study on the above three inference models will be given in the following section. The experimental results confirm the significant superiority of the proposed TF/IVF context sensitive model.

5 Experiments

In this section, we evaluate our three inference models and two other existing methods for tag expansion on a collection of real-life images downloaded from Flickr. An comprehensive performance study on different models is provided.

5.1 Experimental Setup

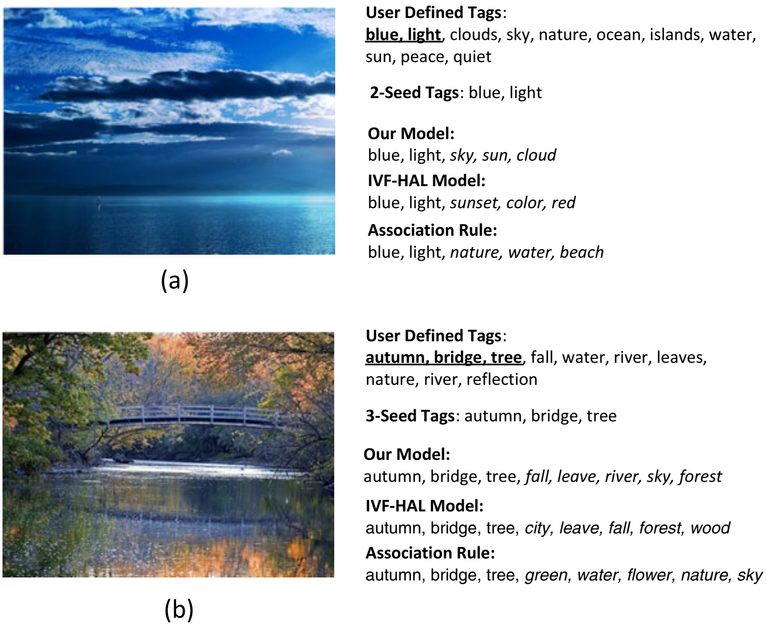
Image Dataset. The image corpus is constructed by performing keyword search on Flickr with "water", "nature", "outdoor", "river", and "environment". A total number of 97,622 different images[1] with 669,470 tags are collected after stemming and removing stop words (e.g.,"the"), camera models (e.g., "Canon"), numbers, and some other insignificant words. The number of unique tags is 14,785. A separate testing dataset of the experiment is also collected from Flickr on the same keywords with time stamp from 10/30/2008 to 04/01/2009. It is pretty hard to get real tagging ground truth. In our assumption, images with larger number of tags are more likely to be well-tagged, where these tags are further considered as the tagging ground truth in the evaluation. Thus we removing the images which contain less than 5 tags. After the preprocessing, we get the testing image set with 4,419 different images.

Evaluation Strategy. For each testing image, its first 1 to 4 tags are picked as the seed tags, based on which the tag expansion is performed by applying different methods. The reason we choose the first few tags as the seed is that we assume Flickr users are used to input closely relevant tags first to an image. Take the images in Figure 2 as the example. The initial tags defined by users of the image (a) are "blue", "light", "clouds","sky", etc. When setting the seed tags as the first two, "blue" and "light", different models give different top 3 recommended tags as the expansion to the seeds tags. For the image (b), we use 3 seed tags and top 5 recommended tags are used for expansion. We assume the initial user defined tags are ground truth, based on which the precision can be calculated for performance evaluation. For the image (a), our model provides all five tags correctly compared with the user defined tags, resulting in a precision@5 = 100%, while precision(IVF-HAL)@5=40% and precision(AR)@5=80%. Top K=8, 10, 15 and 20 tags recommended by different models are evaluated by comparing the corresponding precisions (i.e., precision@K).

5.2 Experimental Results

We show the comprehensive experimental results of two widely used methods including association rule mining and language modeling, and three inference models proposed in the paper in Figure 3.

[1] Uploaded to Flickr from 04/30/2009 to 09/15/2011.



User Defined Tags:
blue, light, clouds, sky, nature, ocean, islands, water, sun, peace, quiet

2-Seed Tags: blue, light

Our Model:
blue, light, *sky, sun, cloud*

IVF-HAL Model:
blue, light, *sunset, color, red*

Association Rule:
blue, light, *nature, water, beach*

(a)

User Defined Tags:
autumn, bridge, tree, fall, water, river, leaves, nature, river, reflection

3-Seed Tags: autumn, bridge, tree

Our Model:
autumn, bridge, tree, *fall, leave, river, sky, forest*

IVF-HAL Model:
autumn, bridge, tree, *city, leave, fall, forest, wood*

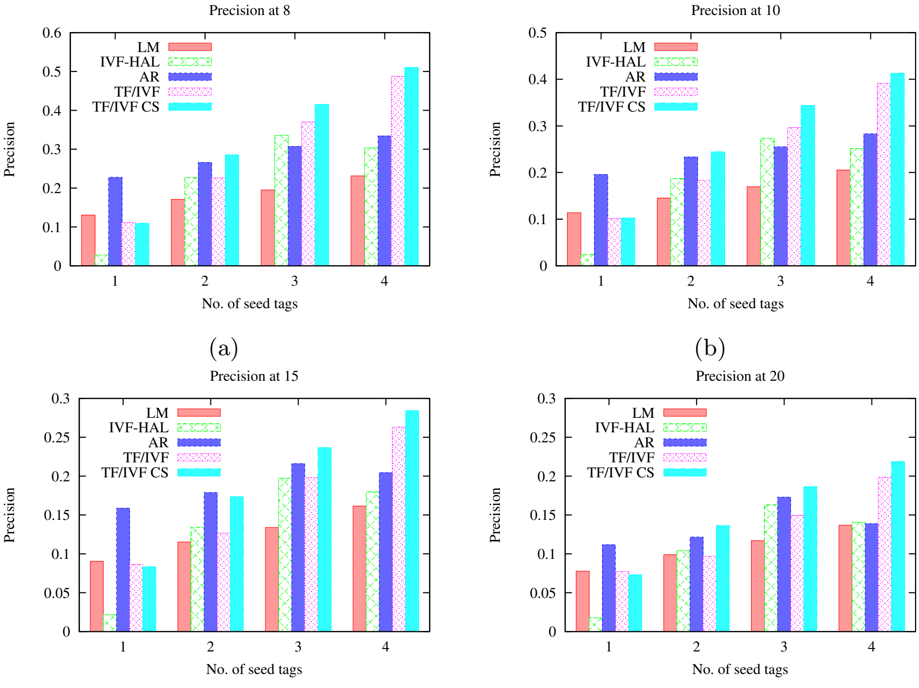
Association Rule:
autumn, bridge, tree, *green, water, flower, nature, sky*

(b)

Fig. 2. An example of tag expansion

Association Rules (AR). Association rules have been investigated by Heymann et al. [11]. The general idea is that if two tags are often used together for image tagging, an explicit strong relationship exists between them. In the experiment, the *min_sup* is set to be 0.00488, and thus 262,555 frequent item sets were derived. Unlike all the other four methods which rank the whole vocabulary according to specified similarity measure and the top K tags are selected as the results, the recommended tags from AR are derived from strong association rules, which makes it possible that AR model is not able to recommend any tags by giving a set of seed tags. As observe from Figure 3, with the increase of the number of seed tags, the precision of AR is dropping when required to recommend 15 or 20 tags for the given seed tags. It is difficult for AR model to discover a large group of tags which have strong associations with each other.

Language Modeling. Tag expansion is similar to query expansion in information retrieval to some extent. Thus, the method of query expansion with term relationships via language modeling [4] is also implemented and compared in our experiment. The performance of this method is the worst among all compared methods, which demonstrates the selected query expansion method is not suitable for tags. In [4], this query expansion is only used to expand the query sentence, which will be later used as the index to select appropriate documents. Expect for query model, there is also a document model in their case. Besides, objective of their query expansion is to generate a better result of documents



(a)

(b)

(c)

(d)

Fig. 3. Comparisons of precision

search. Since we only take the query expansion part of their method (not the entire algorithm), the suboptimal performance could be understandable.

TF/IVF Inference Model. Surprisingly, this method has relatively good performance, especially when the expanded tags are no more than 10 and the seed tags are more than 2. In this method, the large-frequency tags are smoothing by IVF and priority of tightly related tags are increased by weight of the tag in the concept vector of the combined concept. However, a critical problem of the method is if two tags never be co-occurrence but they have very similar context, they are not very likely to be recommended by TF/IVF model. This leads to the need of a method which takes both co-occurrence and similar context into consideration, which is exactly the movitation of our work.

IVF-HAL Inference Model. Based on the original HAL-based information flow [16], an improved information flow algorithm is proposed and used in the IVF-HAL inference model [17] to overcome the problem brought by highly frequent yet less informative words. Degree of each tag is ranked in descending order, where top K tags are returned as expanded tags. Compared with the original HAL model[2], IVF-HAL successfully decreases the weights of generic words with high frequencies when calculating the degree between the combined concept of seed tags and the candidate tags. Observed from Figure 3, the precision of IVF-HAL model is not as good as we expected compared with TF/IVF model. The possible reason is that the image corpus we generated for conceptual space construction is not comprehensive. It contains only few topics according to our keywords setting. The case of the tags with low co-occurrence frequency yet high context similarity is not often. Thus the advantage of IVF-HAL model cannot be shown. However, when we use two seed tags, the precision of IVF-HAL model is always better than TF/IVF model. The parameters used in the model are set as $l_1 = 0.5$, $l_2 = 0.3$, $\alpha = 2$ and $\theta = 0$.

TF/IVF Context Sensitive Inference Model. In the proposed TF/IVF CS model, there are three aspects taken into account, where TF is to increase the rank of highly co-occurring tags, IVF is to alleviate the effect of high frequency tags and *degree* is to take the similarity of context as a judging standard too. As confirmed by the experiment results in Figure 3, the proposed model is the one significantly outperforms other models. Given two seed tags and K=8, the relative precision increases of TF/IVF CS model on AR, LM, TF and IVF-HAL are 35.4%, 113%, 12% and 23.5% respectively. Given three seed tags and K=8, the relative precision increases are 45.9%, 102%, 5.5% and 64%. Compared with TF/IVF model, the effort of *degree* (i.e., context information) is shown by the precision improvement. The parameters used in the model are set as $l_1 = 0.5$, $l_2 = 0.3$, $\alpha = 2$ and $\theta = 0$.

5.3 User-Involved Assessment

We also perform a manual assessment experiment so as to evaluate further on AR, IVF-HAL model and TF/IVF CS model. 20 people are involved to evaluate the performance of different tagging methods on 300 Flickr images (randomly picked from our test image set). Test images along with ten ranked recommended tags are presented to the assessors, who need to pick the tags which they think are appropriate to describe the corresponding image. The experiment results are shown in Figure 4, where we present the precisions of three different models at various K values ($K = 4..10$). The number of seed tags is 3. We can observe that TF/IVF CS model is still superior to other methods. Another interesting observation is that the precision evaluated by the people is higher than by using Flickr image tags as ground truth. The reason is that image tagging is very subjective and the tags used for an image may be synonyms. Take image (a) in Figure 2 as an example. The blue water shown on the image can be annotated as “ocean” or “sea”. In our example, the user chooses “ocean” as one of its tags rather than “sea”. Once the tag expansion models recommend “sea” to the assessors, most likely, it will be accepted as a correct tagging.

[2] Performance comparison between HAL model and IVF-HAL model is not shown in this paper due to the page limit.

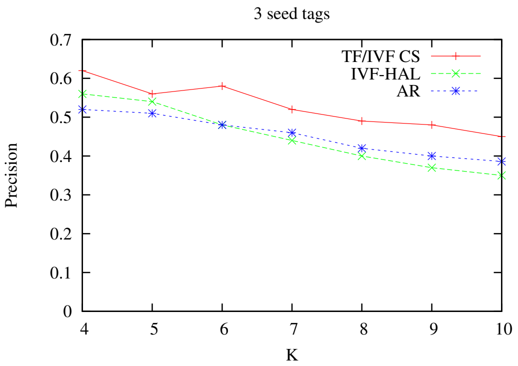


Fig. 4. Comparison of precision with human judgement

6 Conclusion

In this paper, we propose a new tag expansion method by utilizing information inference model for more effective and complete tag expansion. The proposed approach extends Hyperspace Analogue to Language (HAL) model to infer image tags instead of using association rules or latent dirichlet allocations, which takes advantage of information inference for context sensitive tag expansion to overcome the limitation caused by the tag co-occurrence based methods. This approach is able to discover implicit relationship among different tags by analyzing the context of tag appearance. We demonstrate the effectiveness of this proposal with extensive experiments on a large Flickr image dataset, compared with several existing methods.

References

1. Citeulike, http://www.citeulike.org
2. Delicious, http://www.delicious.com
3. Flickr, http://www.flickr.com
4. Bai, J., Song, D., Bruza, P., Nie, J.-Y., Cao, G.: Query expansion using term relationships in language models for information retrieval. In: CIKM, pp. 688–695 (2005)
5. Barwise, J., Seligman, J.: Information Flow: The Logic of Distributed Systems. Cambridge University Press (1997)
6. Burgess, C., Livesay, K., Lund, K.: Explorations in context space: Words, sentences, discourse. Discourse Processes 25(2/3), 211–257 (1998)
7. Burgess, C., Lund, K.: Modeling parsing constraints with high-dimensional context space. Language and Cognitive Processes 12(2/3), 177–210 (1997)
8. Datta, R., Ge, W., Li, J., Wang, J.Z.: Toward bridging the annotation-retrieval gap in image search. IEEE MultiMedia 14(3), 24–35 (2007)
9. Gärdenfors, P.: Conceptual Spaces: The Geometry of Thought. MIT Press (2000)

10. Golder, S.A., Huberman, B.A.: Usage patterns of collaborative tagging systems. J. Information Science 32(2), 198–208 (2006)
11. Heymann, P., Ramage, D., Garcia-Molina, H.: Social tag prediction. In: SIGIR, pp. 531–538 (2008)
12. Krestel, R., Fankhauser, P., Nejdl, W.: Latent dirichlet allocation for tag recommendation. In: Proceedings of the 2009 ACM Conference on Recommender Systems, pp. 61–68 (2009)
13. Marlow, C., Naaman, M., Boyd, D., Davis, M.: Ht06, tagging paper, taxonomy, flickr, academic article, to read. In: Proceedings of the 17th ACM Conference on Hypertext and Hypermedia, pp. 31–40 (2006)
14. Moxley, E., Mei, T., Manjunath, B.S.: Video annotation through search and graph reinforcement mining. IEEE Transactions on Multimedia 12(3), 184–193 (2010)
15. Sigurbjörnsson, B., van Zwol, R.: Flickr tag recommendation based on collective knowledge. In: WWW, pp. 327–336 (2008)
16. Song, D., Bruza, P.: Towards context sensitive information inference. JASIST 54(4), 321–334 (2003)
17. Song, D., Bruza, P., Cole, R.: Concept learning and information inferencing on a high dimensional semantic space. In: Proceedings ACM SIGIR 2004 Workshop on Mathematical/Formal Methods in Information Retrieval (2004)
18. Wang, C., Jing, F., Zhang, L., Zhang, H.: Image annotation refinement using random walk with restarts. In: ACM Multimedia, pp. 647–650 (2006)
19. Wu, L., Yang, L., Yu, N., Hua, X.-S.: Learning to tag. In: WWW, pp. 361–370 (2009)
20. Xu, Z., Fu, Y., Mao, J., Su, D.: Towards the semantic web: Collaborative tag suggestions. In: Proceedings of the Collaborative Web Tagging Workshop at the WWW 2006 (2006)
21. Yang, Y., Huang, Z., Shen, H.T., Zhou, X.: Mining multi-tag association for image tagging. World Wide Web 14(2), 133–156 (2011)

Efficient Subgraph Similarity All-Matching

Gaoping Zhu, Ke Zhu, Wenjie Zhang, Xuemin Lin, and Chuan Xiao

The University of New South Wales, Sydney, NSW, 2052, Australia
{gzhu,kez,zhangw,lxue,chuanx}@cse.unsw.edu.au

Abstract. Being a fundamental problem in managing graph data, subgraph exact all-matching enumerates all isomorphic matches of a query graph q in a large data graph G. The existing techniques focus on pruning non-promising data graph vertices against q. However, the reduction and sharing of intermediate matches have not received adequate attention. These two issues become more critical on subgraph similarity all-matching due to the (possibly) massive number of intermediate matches. This paper studies the problem of efficient subgraph similarity all-matching by developing a novel query processing framework. We propose to effectively decompose a query graph into a hierarchical structure with the aim to minimize the number of intermediate matches and share intermediate matches. Novel techniques are then developed to estimate the number of intermediate matches, efficiently merge the intermediate matches, and generate efficient query execution plans. Experimental on real and synthetic datasets show that our approach outperforms the state-of-the-art approach for orders of magnitude.

1 Introduction

Graphs have been prevalently used in many applications for modeling complex data such as protein interaction networks (i.e., Bio-informatics), chemical compounds (i.e., Chem-informatics), social networks (i.e., Web), etc. Significant research efforts have been made towards many fundamental problems in managing graph data. The problem of *subgraph exact all-matching* is to enumerate all the exact matches (subgraph isomorphism mappings) of a query graph q in a large data graph G. This problem is of great importance in discovering graph structures and well studied in many previous works [16,18].

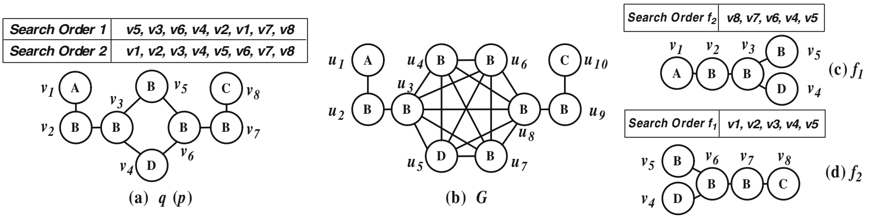
With the explosion of graph data, noisy or inconsistent data are unavoidable in many applications, while query graphs may also be noisy due to erroneous input. Consequently, subgraph exact all-matching may fail to find any exact matches and *subgraph similarity all-matching* is thus strongly demanded in such cases for approximate matches.

This paper studies efficient subgraph similarity all-matching; that is, to enumerate all *similarity matches* of a query graph q in a large data graph G by allowing at most δ missing edges (to be formally defined in Section 2). This problem stems from many applications. For example, in Bio-informatics, we can model protein interaction networks as graphs with proteins and interactions as vertices and edges, respectively. Given a noisy pathway query, our problem can

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 455–469, 2012.
© Springer-Verlag Berlin Heidelberg 2012

return useful similarity matches in the network while no results can be found by subgraph exact all-matching. More applications can be found in [17].

Among all previous works on subgraph similarity matching [7,6,1,8,11,12,17], SAPPER [17] is the only one to enumerate all the similarity matches. It adopts the *enumerate-and-search* paradigm, which firstly identify all *feasible patterns* p (connected subgraphs of q missing at most δ edges of q) and then conducts subgraph exact all-matching to generate the final results. Although straight-forward to implement, the performance of the paradigm drops drastically when the number of intermediate matches is large, which is not uncommon in enumerating all matches. Hence, it is desirable to effectively (1) minimize the number of intermediate matches; and (2) share the intermediate matches to avoid redundant computation. We identify that (1) effective search order and (2) effective query decomposition are the keys to above two issues.



Search Order 1	v5, v3, v6, v4, v2, v1, v7, v8
Search Order 2	v1, v2, v3, v4, v5, v6, v7, v8

Search Order f_2	v8, v7, v6, v4, v5

Search Order f_1	v1, v2, v3, v4, v5

Fig. 1. Effective Search Order and Query Decomposition

Effective Search Order. Consider the query graph q in Figure 1(a) itself as a feasible pattern p, SAPPER conducts subgraph exact all-matching on p against the data graph G in Figure 1(b) by a *depth-first search* [3,9]. The two search orders in Figure 1(a) will encounter 350 and 47 intermediate matches, respectively. Hence, it is important to reduce the intermediate match number by using effective search order.

Effective Query Decomposition. Regarding the feasible pattern p in Figure 1(a) and its two decomposed fragments f_1 and f_2 in Figure 1(c)-(d). According to the search orders in Figure 1(c)-(d), enumerating the exact matches of f_1 and f_2 in G yields a total of 8 intermediate matches for f_1 (including 4 whole matches of f_1) and 8 for f_2 (including 4 whole matches of f_2). In merging the whole matches of f_1 with those of f_2, we only produce at most 16 more intermediate matches for p. Hence, the query decomposition further reduce the number to at most 32 intermediate matches. Moreover, we will show in Section 3 that we can also 'share' the computation cost of intermediate matches by query decomposition.

To the best of our knowledge, this is the first work to propose reducing the number of intermediate matches and sharing the computation of intermediate matches. The main contributions of this paper can be summarized as follows.

1. We propose a novel hierarchical framework DecQ to efficiently conduct subgraph similarity all-matching by decomposing the query into a set of unit sub-queries. We first compute the results of sub-queries (local matching) and then combine them to obtain the final results (global matching).

2. We propose a novel model to estimate the number of intermediate matches produced in local matching and then develop effective search order for sub-queries. In global matching, we develop an efficient merge-and-validation algorithm to combine the results of sub-queries by exploiting the computation sharing among overlapping feasible patterns.
3. We develop efficient heuristic algorithm to generate effective query decomposition for the sharing of intermediate matches.
4. We conducts extensive experiments on both real and synthetic datasets, which demonstrates that our approach outperforms the state-of-the-art approach by up to four orders of magnitude in terms of query response time.

Organizations. We organize the rest of this paper as follows. Section 2 presents important definitions and formalizes the problem. Section 3 proposes our hierarchical querying framework DecQ. Section 4 presents our local matching algorithm, while our global matching algorithm and effective query decomposition are studied in Section 5. Section 6 reports the experimental evaluation. Section 7 surveys related work and Section 8 concludes the paper.

2 Preliminaries

This paper studies *connected, vertex-labeled simple* graphs. A simple graph is an *undirected* graph with neither self-loops nor multiple edges. Without loss of generality, our approach can be easily extended to directed and/or edge-labeled graphs. Given a set Σ_V of labels, a graph g is defined as a triplet $(V(g), E(g), l)$ where $V(g)$ and $E(g) \subseteq V(g) \times V(g)$ are the set of vertices and undirected edges. If an edge is incident on $u, v \in V(g)$, $(u, v) \in E(g)$. The label function $l : V(g) \rightarrow \Sigma_V$ assigns a label $l(v)$ to each vertex $v \in V(g)$.

2.1 Problem Statement

Definition 1 (Subgraph Isomorphism Mapping). *Given two graphs $g = (V, E, l)$ and $g' = (V', E', l')$, a subgraph isomorphism mapping from g to g' is an injective function $f : V \rightarrow V'$ such that (1) $\forall v \in V$, $f(v) \in V', l(v) = l'(f(v))$; (2) $\forall (u, v) \in E$, $(f(u), f(v)) \in E'$.*

Given a subgraph isomorphism mapping from g to g', g is a *subgraph* of g' (g' is a *supergraph* of g), denoted by $g \subseteq g'$. We next define the graph edit distance.

Definition 2 (Graph Edit Distance). *Given two graphs g_1 and g_2, the* graph edit distance *$GED(g_1, g_2)$ from g_1 to g_2 is the minimum number of inserted edges required to transform g_1 to g_2.*

Note that: (1) The edit distance model is not symmetric. If g_1 cannot be transformed to g_2 by edge insertion, $GED(g_1, g_2) = +\infty$. (2) To control the number of similar graphs, we disallow label mismatch or vertex mismatch in the model. From now on, we abbreviate a query graph to a *query*. We next define the feasible pattern of q and similarity matches.

Definition 3 (Feasible Pattern Under δ). *Given a query q and a threshold δ, a feasible pattern of q under δ is a connected subgraph p of q such that $GED(p, q) \leq \delta$. The feasible pattern set $FP(q, \delta)$ consists of all the feasible patterns of q under δ.*

Definition 4 (Similarity Match). *Given a query q, a data graph G and a threshold δ, a similarity match of q in G is a subgraph isomorphism mapping from a feasible pattern p of q to G.*

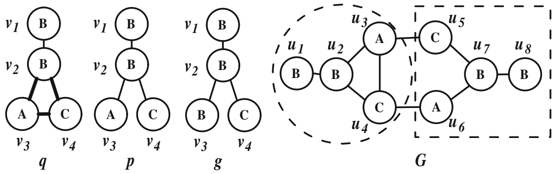


Fig. 2. Subgraph Similarity Matching

Example 1. Regarding Figure 2, assume $\delta = 1$. $p \in FP(q, 1)$ because $GED(p, q) = 1$. $GED(q, p) = +\infty$ as we cannot transform q to p by edge insertion. $GED(g, q) = +\infty$ as the vertex v_3 of q can not be mapped into g. $FP(q, 1)$ contains 4 feasible patterns (q and other 3 feasible patterns by deleting any bold edge in q). Both the two bounded matches in G are similarity matches of q. The circled one is also an exact match of q.

Problem Statement. Given a query q, a data graph G and a threshold δ, subgraph similarity all-matching returns a set S_q consisting of all the similarity matches of q in G.

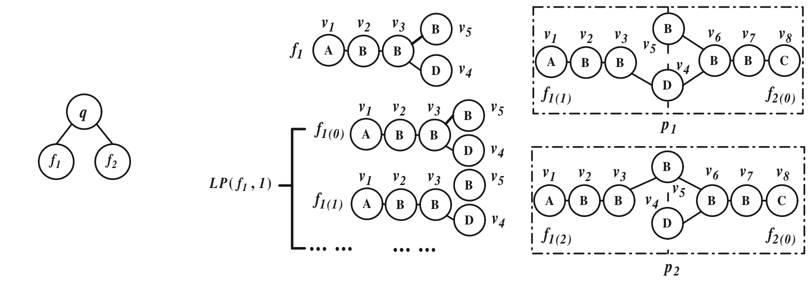
Note that exact subgraph matching is a special case of subgraph similarity matching where $\delta = 0$. Let M_p denote the exact match set of each feasible pattern p. It is immediate that $S_q = \{ M_p | p \in FP(q, \delta) \}$.

3 A Hierarchical Framework

In this section, we propose a novel, three-phase framework DecQ for efficiently processing subgraph similarity all-matching. We summarize it as follows.

Phase 1: Query Decomposition. We decompose the query q into a hierarchical structure (Q, T) which implies a query execution plan. Here, Q is a set of connected, edge-disjoint subgraphs f of q called *fragments* and $\bigcup_{f \in Q} E(f) = E(q)$. Here, Q is also called an *edge-disjoint fragment* cover of q. T is a binary *decomposition tree* whose leaves correspond to all fragments in Q. Each internal node N in T represents a connected subgraph g of q, which can be further decomposed into two edge-disjoint subgraphs g_1 and g_2 residing on the two children. As to the query q in Figure 1, we can decompose q into $Q = \{ f_1, f_2 \}$ and obtain the decomposition tree as in Figure 3(a).

Phase 2: Local Matching. For each fragment $f \in Q$, we first compute its *local pattern set* $LP(f, \delta)$ consists of all local patterns f' (subgraphs of f missing at most δ edges of f). By using depth-first search, we compute the exact



(a) *query decompostion* (b) *local matching* (c) *global matching*

Fig. 3. Our Framework

matches $M_{f'}$, which are called *local matches* (similarity matches of f). For the completeness of final results, we allow f' to be disconnected and compute the exact matches of each component of f' in such case. In Figure 3(b), we have to compute the exact matches for all local patterns including $f_{1(0)}$ and $f_{1(1)}$.

Phase 3 : Global Matching. To distinguish the terms used in local matching, we call the feasible pattern p a *global pattern* and $FP(g, \delta)$ the *global pattern set*, respectively. Given $Q = \{ f_1, ..., f_m \}$, a global pattern p can be assembled from a set of local patterns $\{ f'_1, ...f'_m \}$ such that each $f'_i \in LP(f_i, \delta)$. In global matching, we merge these intermediate matches (local matches) $M_{f'_1}, ..., M_{f'_n}$ to obtain the exact match set M_p of p. Such exact matches of p are similarity matches of q and called *global matches.* Note that the query decomposition provides us an opportunity to share the computation cost of intermediate matches among various global patterns. After all global patterns have been processed, $S_q = \{ M_p | p \in FP(q, \delta) \}$ of q is returned.

For the query q in Figure 1(a), let $\delta = 1$. $p_1 = \{f_{1(1)}, f_{2(0)}\}$ and $p_2 = \{f_{1(2)}, f_{2(0)}\}$ in Figure 3(c)-(d) are two global patterns of q assembled from two local patterns. As p_1 and p_2 share $f_{2(0)}$, we only need to compute the local matches $M_{f_{2(0)}}$ once and share them in the global matching of p_1 and p_2.

4 Local Matching Algorithm

In this section, we propose a model to estimate the number of intermediate matches produced in local matching. We prove that problem of finding the optimal search order with minimized number of estimated intermediate matches is NP-hard and then develop effective search order to reduce the number of intermediate matches.

4.1 Estimating Intermediate Matches

Given a local pattern $f' \in LP(f, \delta)$ and the data graph G, assume a depth-first search algorithm A iteratively searches mappings for each $v \in V(f')$. The number of intermediate matches $|I_{f'}|$ produced in A varies greatly on different search orders employed by A. Although we can not obtain either $|I_{f'}|$ or $|M_{f'}|$ without applying A on f', we propose a novel model to estimate both of them.

For each vertex v (edge e) in a local pattern f', let $M(v)$ ($M(e)$) be the set of its vertex (edge) mappings in G. For each edge $(u, v) \in E(f')$, given any $u' \in M(u)$ and $v' \in M(v)$, the probability that there is an edge $(u', v') \in E(G)$ can be captured by Equation(1).

$$\theta(e) = \begin{cases} \frac{|M(e)|}{|M(v)| \times |M(u)|} & l(u) \neq l(v) \\ \frac{|M(e)|}{|M(v)| \times (|M(u)| - 1)} & l(u) = l(v) \end{cases} \tag{1}$$

Given a search order on $V(f')$ according to which algorithm A searches vertex mappings, let $ig(f', k)$ be the subgraph of f' induced by the first k vertices in $V(f')$. We estimate $|M_{f'}|$ and $|I_{f'}|$ by Equation(2) and (3). Particularly, $|I_{f'}|$ is the summation of $|M_{ig(f',k)}|$ for each resulted induced subgraph $ig(f', k)$ along the search order.

$$\mathcal{E}(M_{f'}) = \prod_{v \in V(f')} |M(v)| \times \prod_{e \in E(f')} \theta(e) \tag{2}$$

$$\mathcal{E}(I_{f'}) = \sum_{i=1}^{|V(f')-1|} \mathcal{E}(M_{ig(f',k)}) \tag{3}$$

Example 2. Consider the fragment f1 and data graph G in Figure 1(c) and (b), assume we only consider vertex label for matching vertices and edges. Figure 4 summarizes M_v and $\theta(e)$ for each $v \in V(f_1)$ and $e \in E(f_1)$. Let the search order on $V(f_1)$ be an ascending order on vertex ID. By Equation 2, $\mathcal{E}(M_{f_1}) = 7^3 \times 1 \times 1 \times (\frac{1}{7}) \times (\frac{5}{7}) \times (\frac{24}{42})^2 = 11.4$. By Equation 3, $\mathcal{E}(I_{f_1}) = 1 + 1 + 4 + 2.8 = 8.8$.

v	$\|M(v)\|$
v_1	1
v_4	1
other v	7

e	$\theta(e)$
(v_1, v_2)	$\frac{1}{1 \times 7} = 0.143$
(v_3, v_4)	$\frac{5}{1 \times 7} = 0.714$
other e	$\frac{24}{7 \times 6} = 0.571$

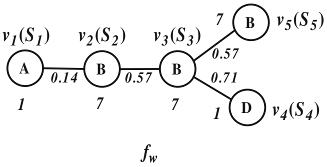


Fig. 4. Effective Search Order

Theorem 1. *Given any fragment f, the problem of finding the optimal search order with minimum estimated number of intermediate matches is NP-hard.*

Proof Sketch. It is immediate that Theorem 1 can be proved by reduction from maximum clique. Due to the interest of space, we omit the proof here.

4.2 Effective Search Order

Seeing the difficulty in finding the optimal search order, we proposes a heuristic algorithm to obtain an effective search order. Given a local pattern f', we first transform it into a weighted graph f_w such that (1) $\forall v \in V(f_w)$, $w(v) = |M(v)|$; (2) $\forall e \in E(f_w)$, $w(e) = \theta(e)$. For any subgraph g of f_w, $\mathcal{E}(M_g)$ equals the produce of all vertex and edge weights on g. Our algorithm iteratively selects a vertex $v \in V(f_w)$ into the current search order V, which results in a new subgraph g of f' induced by V. In each iteration, we greedily select the vertex such that the resulted g has minimum estimation $\mathcal{E}(M_g)$. Consequently, the algorithm aims to to minimize $\mathcal{E}(M_g)$ for each resulted induced subgraph along the search order. Our algorithm GenOrder is outlined in Algorithm 1.

Algorithm 1: GenOrder (f_w)

```
Input  : $f_w$: a weighted graph;
Output : $V$: an ordered set of vertices, initially an empty set;
Pick any $v' \in V(f_w)$ s.t. $\nexists\ v \in V(f_w) \wedge w(v) < w(v')$;
$V := V \bigcup \{ v' \}, V(f_w) := V(f_w) - \{ v' \}$;
while $V(f_w) \neq \emptyset$ do
    Pick any $v \in V(f_w)$ such that $\mathcal{E}(M_g)$ is minimized for the subgraph $g$ of $f'$
    induced by $V \bigcup \{ v \}$;
    $V := V \bigcup \{ v' \}, V(f_w) := V(f_w) - \{ v' \}$;
return $V$;
```

Complexity Analysis. Clearly, GenOrder needs $O(|V(f_w)|)$ iterations and runs in $O(|V(f_w)||E(f_w)|)$. The space requirement is $O(|V(f_w)| + |E(f_w)|)$.

Example 3. Regarding the weighted graph f_w in Figure 4 transformed from f_1 in Figure 1(c), we weight all the vertices and edges following the left table. GenOrder will select S_i as the i-th vertex in the search order.

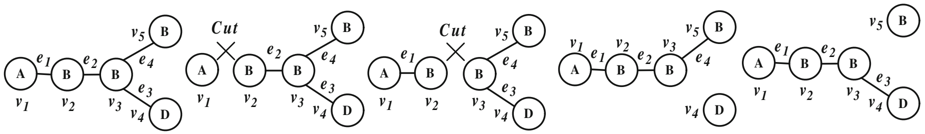
4.3 Efficient Local Matching

Enumerating Local Patterns. To conduct local matching, we must compute the local pattern set $LP(f, \delta)$ of each fragment $f \in Q$. Unlike the connected global patterns, any subgraph f' (connected or disconnected) of f missing no more than δ edges is a potential local pattern. This is because the connected components of f' may be bridges by other local patterns to form a global pattern. Given a *total order* on $E(f)$, for a subgraph f' of f missing at most δ edges, *key* of f' is defined as a set of 'ordered edges' missed in f; that is, $Key(f') = \{ e | e \in E(f) \wedge e \notin E(f') \}$. Below, we define a lexicographic order on $Key(f')$.

Definition 5 (Lexicographic Order). *Assume* $Key(f_a) = \{e^a_1, ..., e^a_k\}$ *and* $Key(f_b) = \{e^b_1, ..., e^b_l\}$ *represent two subgraphs* f_a*,* f_b *of a fragment* f*.* $Key(f_a)$ *¡* $Key(f_b)$ *if and only if, (1)* $Key(f_a) = \emptyset$*; or (2)* $\exists j \in [1, min(k, l)]$ *s.t.* $\forall i \in [0, j], e^a_i = e^b_i \wedge e^a_{j+1} < e^b_{j+1}$*; or (3)* $k < l$ *and* $\forall i \in [1, k], e^b_i = e^b_i$*.*

We enumerate all subgraphs f' in ascending order on $Key(f')$ and insert f' into $LP(f, \delta)$ if $Key(f')$ is not an edge cut of q. This is because f' can not be combined with other local patterns to for a connected global pattern. For disconnected local patterns f', we remove all isolated vertices from f' because these vertices must be presented in other fragments in Q.

Computing Local Matches. We extend the subgraph isomorphism test algorithm QuickSI [9] and adopt our effective search order to compute all local matches. For local matches, we maintain $LP(f, \delta)$ in a *pattern table* $T(f)$ where each record $(Key(f'), M_{f'}) \in T(f)$ represents a local pattern f' and its exact match set. If f' is disconnected into a set $\{ c_i | 1 \leq i \leq n \}$ of n connected components, $M_{f'}$ is replaced with the exact match sets M_{c_i} of each c_i.



(a) $Key(a) = \varnothing$ (b) $Key(b) = \{e_1\}$ (c) $Key(c) = \{e_2\}$ (d) $Key(d) = \{e_3\}$ (e) $Key(e) = \{e_4\}$

Fig. 5. Enumerating Local Patterns

Example 4. Regarding the fragment f_1 in Figure 1, assume $\delta = 1$. Figure 5$(a) - (e)$ show all 5 subgraphs of f_1 missing at most one edge. Since $Key(b)$ and $Key(c)$ are two edge cuts, $LP(f_1, \delta) = \{ (a), (d), (e) \}$. Finally, we remove v_4 and v_5 from (d) and (e) as they are single-vertex components.

5 Global Matching Algorithm

In this section, we propose an efficient merge-and-validation global matching algorithm, which shares the computation cost of intermediate matches of various global patterns. Then we develop effective query decomposition technique to maximize the computation sharing.

5.1 Enumerating Global Patterns

Similar as in Section 4, we assign any subgraph q' of q missing at most δ edges with a key $Key(q') = \{ e | e \notin E(q') \wedge e \in E(q) \}$ representing the ordered missing edges. The lexicographic order on $Key(q')$ can be similar defined. We organize all global patterns of q in a *pattern lattice* with with $|E(q)| + 1$ levels. Level-i contains the keys $Key(q')$ of all q' missing i edges in q. On the top and bottom level, we put $Key(q) = \emptyset$ and $E(q)$, respectively. Due to the error threshold δ, we can safely discard all the levels below the δ-th level. For any two subgraphs q_a and q_b from the i-th and $i+1$-th level, if q_a is obtained by removing an edge from q_b, we call q_a a child of q_b (q_b a parent of q_a). We order all the subgraphs q' on level-i in ascending lexicographic order on $Key(q')$. Clearly, if $Key(q')$ is an edge cut of q, q' is disconnected and thus not a global pattern; otherwise, q' is a global pattern which must fall in one of the following two categories.

- If $Key(q'')$ is an edge cut of q for all the children q'' of q', or q' has no children, we call such q' a **minimal pattern**.
- If $Key(q'')$ is not an edge cut of q for some child q'' of q', we call such q' a **non-minimal pattern**.

Example 5. Regarding the query q in Figure 6, we depict its pattern lattice for $\delta = 2$ with all the subgraphs represented by their keys. We bound all subgraphs q' with rectangles if $Key(q')$ is an edge cut of q. All the minimal patterns are circled, while all the global patterns above level-2 are non-minimal patterns.

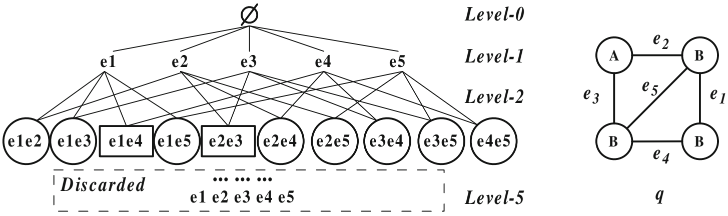


Fig. 6. Pattern Lattice

Our merge-and-validation matching algorithm is presented in Algorithm 2 which traverse the pattern lattice level by level, according to the order of $Key(q')$. The algorithm processes minimal patterns with sharing-aware merge, while efficient edge validation is adopted to process non-minimal patterns. We give details on the sharing-aware merge and edge validation in the following two subsections.

```
Algorithm 2: GlobalMatch $(q, G, L, \delta)$
Input  : $q$: a query; $G$: a data graph; $L$: the pattern lattice; $\delta$: the threshold;
Output : $S_q$: similarity match set of $q$;
for $i := \delta$ to 0 do
    for each $q'$ in ascending order on level-$i$ do
        if $Key(q')$ is an edge cut then continue;
        if $q'$ is a minimal pattern then
            Compute $M_{q'}$ by sharing-aware merge;
        else
            Compute $M_{q'}$ by edge validation;
        $S_q := S_q \bigcup \{ M_{q'} \}$;
return $S_q$;
```

5.2 Matching Minimal Patterns

We compute the exact matches M_p of minimal patterns p by merging the intermediate matches under according to the decomposition tree T in the query execution plan. Note that any internal node N in T indicates a connected subgraph g of q, which are further decomposed into two edge-disjoint subgraphs residing on the two child nodes L and R of N. Let $N.g$ be the subgraph represented by N. Let $J = V(L.g) \bigcap V(R.g)$ be the common vertices of $L.g$ and $R.g$. We compute $M_{N.g}$ by equi-joining $M_{L.g}$ and $M_{R.g}$ on J. In practice, we adopt hash join to perform the task. Generally, following (Q, T), we first decompose p along T in a top-down fashion to retrieve the decomposed local patterns corresponding to fragments in Q and then recursively merge the intermediate matches (local matches) to compute M_p.

Sharing Intermediate Matches. Given two minimal patterns p and p', if they share common intermediate patterns, we can share the merge cost of them. Such intermediate patterns are either local patterns of the merge results of a set of local patterns. We create an intermediate pattern table $T(N)$ on each internal node N. For each newly encountered intermediate pattern $N.g'$ on N merged from some local patterns, we insert a tuple $(N.g', M_{N.g'})$ into $T(N)$. Consequently, if two patterns p and p' share $N.g'$, we can share the pre-computed $M_{N.g'}$ to avoid redundant merge cost.

Maintain Disconnected Intermediate Matches. Due to the possibly disconnected local patterns, an intermediate pattern $N.g'$ may not be connected. In such case, we maintain the exact matches for each of its component and delay their merge until a 'bridge' intermediate pattern links them at a later stage.

5.3 Matching Non-minimal Patterns

For a non-minimal patterns p at level-i, any child pattern p' of p at level-$i+1$ only miss one edge in p. According to Definition 2, any exact match of p must be an exact match of p'. Thus, we only need to conduct edge validation on $M_{p'}$ for computing M_p. We pick the child p' of p with minimum $|M_{p'}|$ and check each exact match $\mathcal{F} \in M_{p'}$ to see if the extra edge in p exists in $\mathcal{F}$. If so, $\mathcal{F}$ is also an exact match of M_p.

5.4 Effective Query Decomposition

Given a global pattern p decomposed into a set of local patterns, the computation cost of M_p contains (1) the search cost of local patterns and (2) the merge cost of intermediate patterns. The search cost of a local pattern f' can be evaluated by $\mathcal{E}(M_{f'}) + \mathcal{E}(I_{f'})$, while the merge cost of an intermediate pattern g' can also be evaluated by $\mathcal{E}(M_{g'})$. Equation(3) and (2) can be used to calculate the cost.

However, it is expensive to generate an optimal decomposition for each $p \in FP(q, \delta)$ is expensive as there are too many p and possible decompositions to consider. Hence, we propose *recursive bisection* to generate a uniform decomposition for all global patterns by considering q only.

Recursive Bisection. The recursive bisection works as follows. We initialize an empty query cover Q and a decomposition tree T with only one root node R representing q. The we recursively bisect q and its successive decomposed subgraphs to construct Q and T. For each newly decomposed subgraph g, we build a new leaf node N in T to hold g as a fragment. Consequently, the computation cost on N is simply the search cost of all local patterns g' of g. We the attempt to bisect g into g_1 and g_2, two smaller fragments to reduce the computation cost. The new computation cost contains (1) the search cost of local patterns of g_1 and g_2 and (2) the merge cost of the local patterns of g_1 and g_2.

Equation (4) and (5) estimate computation cost on N before and after the bisection, respectively. Note that we approximate the search cost of all local patterns by the search cost of their corresponding fragments. According to Definition 2, the number of possible local patterns of g is $\alpha_g = \Sigma_{i=0}^{\delta}\binom{|E(g)|}{i}$. Recall that we use $\mathcal{E}(M_g)$ to estimate the merge cost. Similarly, if we decompose g, there are at most α_g intermediate patterns to be merged. Equation 6 gives the cost gain of the bisection.

$$C_a = \alpha_g(\mathcal{E}(I_g) + \mathcal{E}(M_g)) \tag{4}$$

$$C_b = \alpha_{g_1}(\mathcal{E}(M_{g_1}) + \mathcal{E}(I_{g_1})) + \alpha_{g_2}(\mathcal{E}(M_{g_2}) + \mathcal{E}(I_{g_2})) + \alpha(g)(\mathcal{E}(M_g)) \tag{5}$$

$$C_g = \alpha_g \mathcal{E}(I_g) - \alpha_{g_1}(\mathcal{E}(M_{g_1}) + \mathcal{E}(I_{g_1})) - \alpha_{g_2}(\mathcal{E}(M_{g_2}) + \mathcal{E}(I_{g_2})) \tag{6}$$

Since $\mathcal{E}(I_g)$ is fixed with a pre-given search order before bisection, we aim to reduce the search cost of g_1 and g_2. Note that a good bisection should be balanced since both $\mathcal{E}(M_{g_1})$ and $\mathcal{E}(I_{g_2})$ has an exponential growth with the increase of graph size. Hence, our bisection always aims to bisect g into two connected subgraphs with approximately the same size. In experiments, we only bisect g when it yields a positive cost gain. The minimum fragment size is $\delta + 1$.

6 Performance Evaluation

In this section, we report our experimental results and analyses. We obtain the binary code of SAPPER from its authors [17]. All our algorithms are implemented in C++ and compiled with GCC 4.3.2 with -O3 flag. All experiments are conducted on a PC with Intel Xeon 2.40GHz CPU and 4GB memory running Debian 4.1.1-21.

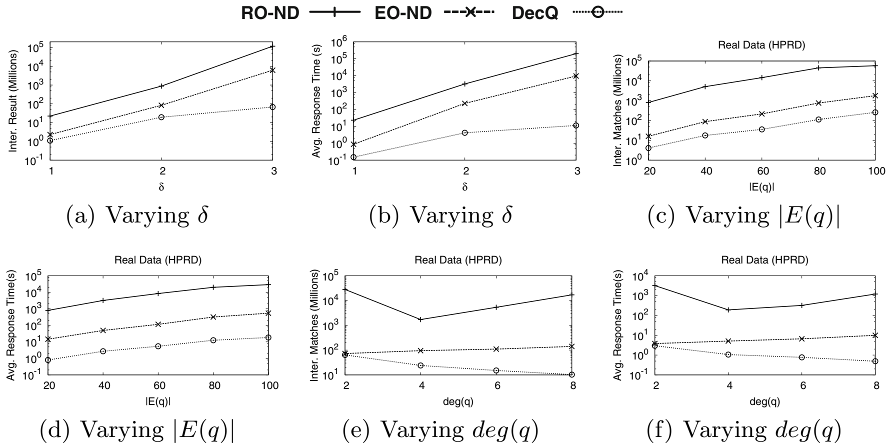
Datasets. Our real dataset is the Human Protein Interaction Network, a popular benchmark (http://www.hprd.org/download) for evaluating subgraph matching and search techniques. The network, denoted G_H, consists of $9,460$ vertices, $37,081$ edges and 307 distinct vertex labels. We adopt G_H to study the efficiency of our proposed algorithms. We generate synthetic data graphs and queries to study the scalability of our proposed algorithms varying data graph settings. Note that the queries are always generated by selecting induced data graphs from the underlying data graphs and we randomly insert $1-3$ 'noisy edges'. All query set contains 100 queries. The synthetic queries are similarly generated as the real queries. We summarize the default parameters of query and data graphs in Table 1. Note that $|V(G)|$, $deg(G)$, and $|\Sigma_V|$ are applicable for synthetic datasets only. The default error threshold δ is 2 unless otherwise specified.

Table 1. Default Values of Parameters

Parameters	$\lvert E(q)\rvert$	avg. $deg(q)$	$\lvert V(G)\rvert$	avg. $deg(G)$	$\lvert\Sigma V\rvert$
Default Values	40	4	5,000	12	100

Evaluated Algorithms. We evaluate the following algorithms in this paper: (1) RO-ND The basic subgraph similarity all-matching algorithm which enumerates feasible patterns of q and searches the exact matches of feasible patterns with random search order; (2) EO-ND The modified RO-ND algorithm equipped with effective search order. (3) DecQ Our proposed algorithm with effective search order and effective query decomposition. (4) SAPPER The algorithm developed in [17]. In order to facilitate the local matching, the indexing technique in [17] is applied on all algorithms to efficiently identify candidate data graph vertices.

6.1 Varying Error Threshold and Query Settings



(a) Varying δ

(b) Varying δ

(c) Varying $|E(q)|$

(d) Varying $|E(q)|$

(e) Varying $deg(q)$

(f) Varying $deg(q)$

Fig. 7. Varying Error Threshold and Query Settings

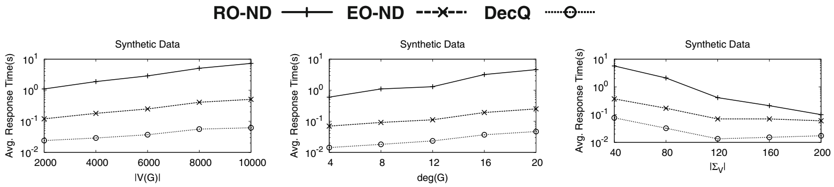
Varying Error Threshold δ. We compare 3 algorithms RO-ND, EO-ND, DecQ on G_H to study the effect of our effective search order and query decomposition. We plot the averaged number of intermediate matches and query response time in Figure 7(a) and 7(b). Clearly, the intermediate matches of RO-ND and EO-ND grow the fastest, while DecQ have decent growths. Thanks to the effective search order, EO-ND produces only up to 1/19 as many intermediate matches as RO-ND does. DecQ outperforms the other algorithms on all δ settings.

The trend of query response time confirms the effectiveness and efficiency of our proposed techniques. All algorithms costs more time for larger δ, while DecQ is the most efficient among the four. When $\delta = 3$, EO-ND is 21 times faster than RO-ND, while DecQ has an additional speed-up over EO-ND for up to 840 times. The gaps between DecQ and other algorithms increase when δ increases because more computation on overlapping global patterns can be shared.

Varying Query Size $|E(q)|$. We next evaluate the effect of query size on G_H. The intermediate matches and the query response time are plotted in Figure 7(c) and 7(d). The intermediate matches and the response time both increases with the query size. The reason is that we have to go deeper in the depth-first search for RO-ND and EO-ND, or decompose q into more fragments for DecQ; yet both our search order and query decomposition are effective over all $|E(q)|$ settings.

Varying Average Query Density $deg(q)$. We then evaluate the effect of query density on G_H and report the results in Figures 7(e) and 7(f). It is interesting that all algorithms exhibit different trends. The response time of RO-ND first decreases and then rebounds, while that of EO-ND almost levels over all density settings. The response time of DecQ keeps decreases when q becomes denser. Same trend is observed on the number of intermediate matches. There are two counteracting factors that affect this result: (1) the number of global patterns increases for denser queries; (2) dense global patterns are less likely to have matches due to the strict topological structure. For RO-ND, the second factor is dominant for small degrees, while the first factor is more significant when $deg(q) > 4$. For EO-ND, the effective search order makes it more efficient to find matches and hence weakened the effect of the first factor. Thanks to the shared intermediate matches, the matching is even faster for DecQ, and hence the second factor dominates.

6.2 Varying Data Graph Settings



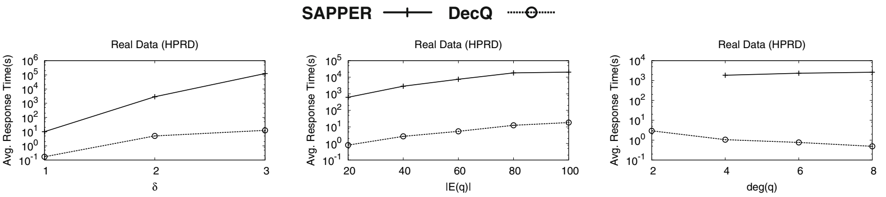
(a) Varying $|E(g)|$ (b) Varying $deg(g)$ (c) Varying $|\Sigma_V|$

Fig. 8. Varying Data Graph Settings

Varying Data Graph Size and Density. We firstly evaluate the effect of data graph size and density on synthetic dataset. The results with varying data graph size and density are reported in Figure 8(a) and 8(b). DecQ achieves 8 times speed-up against EO-ND and 118 times against RO-ND over all graph size settings. DecQ and EO-ND exhibit lower growth rate because the effective search order starts with the most selective vertex, whose number in G does not grow as fast as $|V(G)|$. Similar trend is observed on all density settings which confirms DecQ has better scalability than the other algorithms.

Varying Number of Vertex Labels. We report our results over different $|\Sigma_V|$ settings in Figure 8(c). All algorithms consumes less time when $|\Sigma_V|$ increases. This is because the vertices are rendered more selective and thus leads to few intermediate matches. The response time almost levels for both EO-ND and DecQ when the $|\Sigma_V|$ exceed 120. This is because the selectivity of the most label selective vertices in the search order barely changes when we include more labels.

6.3 Comparison with SAPPER



(a) Response Time (b) Response Time (c) Response Time

Fig. 9. Comparison with SAPPER

We finally compare DecQ with SAPPER by varying δ, $|E(q)|$ and $deg(q)$ on G_H. The results are reported in Figure 9(a) to 9(c). When $\delta = 3$, DecQ is faster than SAPPER by 4 orders of magnitude. We do not report the results of SAPPER on $deg(q) = 2$ since it runs out of all $4GB$ memory in storing intermediate matches. The large gap on response time between two algorithms is witnessed on all experiments. This is mainly due to our effective search order and query decomposition. The reduction and sharing of intermediate matches significantly save the cost for processing highly overlapping global patterns. Note that two algorithm exhibits different trend on query density settings. The main reason is that more global patterns are enumerated for denser queries, while they are more selective and less likely to have matches. Since DecQ shares the computation cost among global patterns, it is less sensitive to the effect of increasing global patterns. Consider both factors, the decreased response time can be explained.

7 Related Work

Many fundamental problems in managing graph data has been extensively studied. These include subgraph exact and similarity all-matching, subgraph/supergraph containment search and subgraph similarity search. On *exact subgraph all-matching*, most studies propose to build efficient index to prune non-promising data graph vertices against the query. [16] develops an indexing technique called GADDI to index nearby discriminative subgraphs as signatures, while shortest path are also adopted in [18] as unit index structure. To handle noisy graph data, [11] studies subgraph similarity all-matching by developing neighborhood-based index structure. [17] on the other hand, transform the problem to subgraph exact all-matching by proposing the enumerate-and-search paradigm.

Subgraph containment search [4,5,9,10,13,14,19,20] and supergraph containment search. [2,15] also attract great research interests. On *subgraph containment search*, [10] proposes the first filtering-verification framework by indexing path-features to filter false results before the expensive verification. [13] improves the filtering power by indexing discriminative graph-features. To further reduce filtering cost and index construction size, [19] and [14] independently propose to adopt tree-features. [9] proposes efficient verification approach to accelerate query processing. On *supergraph containment search*, [2] propose cIndex to select contrast features via query log, while [15] enhances the verification phase by sharing search cost on common subgraphs of data graphs. On *subgraph similarity search*, [12] follows the filtering-verification framework to remove false results by counting the number of missing features. Most recently, [8] proposes efficient verification algorithm and a novel filtering-validation-verification paradigm to process the problem.

8 Conclusions

In this paper, we study the problem of efficient subgraph similarity all-matching. We develop a hierarchical framework DecQ to firstly decompose the query into a set of unit sub-queries and then combine the results of sub-queries for final results. We propose novel intermediate match estimation model and develop heuristic algorithm to generate effective search order for the reduction of intermediate matches. We develop a merge-and-validation algorithm to combine sub-query results by sharing the computation cost of intermediate matches. Our experimental results demonstrate that our proposed approach outperforms the state-of-the-art approaches by up to 4 orders of magnitude in terms of both intermediate match number and query response time.

References

1. Bunke, H., Foggia, P., Guidobaldi, C., Sansone, C., Vento, M.: A comparison of algorithms for maximum common subgraph on randomly connected graphs. In: SSPR/SPR, pp. 123–132 (2002)
2. Chen, C., Yan, X., Yu, P.S., Han, J., Zhang, D.-Q., Gu, X.: Towards graph containment search and indexing. In: VLDB, pp. 926–937 (2007)
3. Cordella, L., Foggia, P., Sansone, C., Vento, M.: An improved algorithm for matching large graphs. In: 3rd Workshop on Graph-based Representations in Pattern Recognition, pp. 149–159 (2001)
4. He, H., Singh, A.K.: Closure-tree: An index structure for graph queries. In: ICDE, p. 38 (2006)
5. Jiang, H., Wang, H., Yu, P.S., Zhou, S.: Gstring: A novel approach for efficient search in graph databases. In: ICDE, pp. 566–575 (2007)
6. Krissinel, E.B., Henrick, K.: Common subgraph isomorphism detection by backtracking search. Softw. Pract. Exper. 34(6), 591–607 (2004)
7. McGregor, J.J.: Backtrack search algorithms and the maximal common subgraph problem. Softw. Pract. Exper. 12(1), 23–34 (1982)
8. Shang, H., Lin, X., Zhang, Y., Yu, J.X., Wang, W.: Connected substructure similarity search. In: SIGMOD, pp. 903–914 (2010)
9. Shang, H., Zhang, Y., Lin, X., Yu, J.X.: Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. PVLDB 1(1), 364–375 (2008)
10. Shasha, D., Wang, J.T.-L., Giugno, R.: Algorithmics and applications of tree and graph searching. In: PODS, pp. 39–52 (2002)
11. Tian, Y., Patel, J.M.: Tale: A tool for approximate large graph matching. In: ICDE, pp. 963–972 (2008)
12. Yan, X., Han, P.S.Y.J.: Substructure similarity search in graph databases. In: SIGMOD, pp. 766–777 (2005)
13. Yan, X., Yu, P.S., Han, J.: Graph indexing: A frequent structure-based approach. In: SIGMOD Conference, pp. 335–346 (2004)
14. Zhang, S., Hu, M., Yang, J.: Treepi: A novel graph indexing method. In: ICDE, pp. 966–975 (2007)
15. Zhang, S., Li, J., Gao, H., Zou, Z.: A novel approach for efficient supergraph query processing on graph databases. In: EDBT, pp. 204–215 (2009)
16. Zhang, S., Li, S., Yang, J.: Gaddi: distance index based subgraph matching in biological networks. In: EDBT, pp. 192–203 (2009)
17. Zhang, S., Yang, J., Jin, W.: Sapper: Subgraph indexing and approximate matching in large graphs. In: VLDB (2010)
18. Zhao, P., Han, J.: On graph query optimization in large networks. PVLDB 3(1), 340–351 (2010)
19. Zhao, P., Yu, J.X., Yu, P.S.: Graph Indexing: Tree + Delta >= Graph. In: VLDB, pp. 938–949 (2007)
20. Zou, L., Chen, L., Yu, J.X., Lu, Y.: A novel spectral coding in a large graph database. In: EDBT, pp. 181–192 (2008)

Efficient Algorithm for Mining Correlated Protein-DNA Binding Cores

Po-Yuen Wong, Tak-Ming Chan, Man-Hon Wong, and Kwong-Sak Leung

Department of Computer Science and Engineering
The Chinese University of Hong Kong
{pywong,tmchan,mhwong,ksleung}@cse.cuhk.edu.hk

Abstract. Correlated protein-DNA interaction (binding cores) between transcription factor (TFs) and transcription factor binding sites (TFBSs) are usually identified by costly 3D structural experiments. To avoid numerous unsuccessful trials, we are motivated to develop a cheap and efficient sequence-based computational method for providing testable novel binding cores with high confidence to accelerate the experiments. Although there are abundant sequence-based motif discovery algorithms, few directly address associating both TF and TFBS core motifs which are both verifiable on 3D structures. In this paper, we formally define the problem of discovering correlated TF-TFBS binding cores, and apply association rule mining techniques over existing real sequence data (TRANSFAC). The proposed algorithm first builds two frequent sequence tree (FS-Tree) structures storing condensed information for association rule mining. Association rules are then generated by depth-first traversal on the structures. FS-Trees have several advantages to support further applications, including efficient calculation of the support and confidence, simple generation of candidate rules, and applicability of effective pruning techniques. As a result, the FS-Trees serve as a useful basis for more general extensions related to biological binding core identification. We tested our algorithm on real sequence data from the biological database TRANSFAC and focus on efficiency comparisons with the recent work employing association rule mining. The rules discovered reveal real TF-TFBS binding cores in independent 3D verifications on Protein Data Bank (PDB).

Keywords: Bioinformatics, association rule mining, sequence, algorithm.

1 Introduction

An important problem in bioinformatics is the identification of binding cores between transcription factors and transcription factor binding sites [14]. A short region of DNA that is called the transcription factor binding site (TFBS) can be bound by proteins called transcription factors (TFs) in a sequence specific manner. The bound TF interacts with other proteins and finally controls (regulates) the expression of a gene. In computational biology, TFs are represented by sequences in lengths of 400–2000 with 20 different letters. TFBSs are represented by short sequences in lengths usually less than 10 with 4 different letters. Sequence-based biological experiments characterizing TFBSs are however in lower resolutions of lengths 20–100. A simplified model of the TF-TFBS interactions are depicted in Figure 1. A TF binds to its corresponding TFBS

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 470–481, 2012.
© Springer-Verlag Berlin Heidelberg 2012

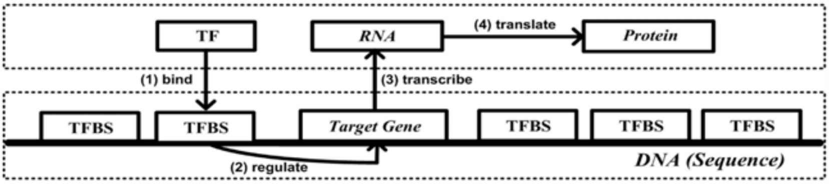


Fig. 1. A simplified model of TF-TFBS interactions

to regulate the gene, i.e. control the transcription rate from the gene to RNA. The transcribed RNA is finally translated into a protein in another process.

Experiments such as DNA footprinting [11], gel electrophoresis [12], and Chromatin immunoprecipitation [18,25] are conducted by scientists to determine whether a TF sequence binds a TFBS sequence. These sequence data are collected in online databases such as TRANSFAC [19]. However, only short regions within several letters of a TF and a TFBS, called binding cores, are critical in the bindings. The resolutions of these experimental results are too low to identify binding cores. Therefore, expensive and time consuming experiments that extract high-resolution 3D structures using X-ray crystallography or nuclear magnetic resonance spectroscopic analysis are conducted to identify them. As huge amounts of time and cost are wasted in conducting unsuccessful trials, we are motivated to develop a cheap and efficient computational method for providing testable candidates of novel binding cores with high confidence to accelerate the 3D structural experiments.

In [15], we have successfully showed the potential of applying association rule mining in predicting TF-TFBS binding cores. The philosophy behind is that as TF-TFBS bindings play an essential role in gene regulation, the patterns of correlated TF-TFBS bindings should be present through the evolution process and the same patterns should appear more than random in different genes and different organisms. Therefore, if a region of a TF exists together with a region of TFBS frequently in the database, the regions are likely to be binding cores. We verified the hypothesis by transforming a biological database into the format of a transaction database and applying an existing algorithm for association rule mining. Although promising results are found, the approach suffers from several drawbacks.

In this paper, we formally define the problem of mining protein-DNA association rules over biological databases. Then, we propose a novel and efficient association rule mining based algorithm using an intermediate data structure called a *Frequent Sequence Tree* (FS-Tree). The FS-Tree has several advantages, including efficient calculation of the support and confidence, simple generation of candidate rules, and applicability of effective pruning techniques. As a result, the FS-Trees serve as a useful basis for more general extensions related to biological binding core identification. Lastly, we evaluate the algorithm on a real biological dataset.

The organization of the rest of the paper is as follows. In Section 2, we review related works. In Section 3, we give formal definitions of the problem. We present the algorithm in Section 4 and evaluate it in Section 5. Lastly, we conclude in Section 6.

2 Related Work

In the computational biology community, the sequence-based pattern discovery [16] problem called motif discovery [7,17,22] has been extensively addressed. The numerous approaches proposed are related to identify binding cores on either TF or TFBS separately from sequence data. However, the existing motif discovery methods do not correlate the potential TF and TFBS binding cores (motifs) and do not provide any direct linkage between them. Even if correct TF and TFBS binding cores (motifs) are discovered separately, they may not be the corresponding counterparts. Therefore, associating potential TF-TFBS binding cores is not only novel but also significant. This can be achieved by exploiting the abundant binding sequence pair available in the biological databases such as TRANSFAC [19].

In the database community, the problem of association rule mining is to find all co-existing itemsets which are both frequently appears independently (support) and dependently (confidence) in a transaction database. The Apriori algorithm [3] is commonly regarded as the classic algorithm. There are extensive literatures on improving and extending Apriori-like algorithms [1,2,4,8,10,13,20,21,27,29]. Recent sequential pattern mining considers the ordered transactions of customers as sequences and finds frequent sequential patterns [5,6,23,26,30]. Sequential pattern for biological sequences is also studied in [28]. The algorithm considers several properties of biological sequences to speed up the mining.

Our problem is fundamentally different from both of the above problems as we are not finding co-existences between items nor only frequent subsequences in the database. Our problem is the mixture of the above two, aiming to find co-existences between frequent subsequences of particularly two different types in the database. Furthermore, we only consider continuous subsequences comprising of letters (items) while a subsequence in sequential pattern mining problems can be discontinuous sets of letters (itemsets). Therefore, the algorithms proposed in the above two areas cannot be applied directly to our problem.

In [15], we applied association rule mining technique over TRANSFAC by transforming the database into the format of a transaction database and employed the *Apriori* algorithm to generate association rules. However, the approach suffers from several limitations: (i) Only association rules between a fixed length of subsequences can be mined; (ii) The size of the transformed database is huge; (iii) Irrelevant association rules which have either TFs or TFBSs on both sides are generated. Therefore, in this paper, we formally define the problem and design an efficient algorithm to overcome these drawbacks with better scalability.

3 Problem Statement

In this section, we formalize the problem of mining protein-DNA association rules over biological databases.

In our problem, we are given a biological database as the input, in which each entry is a record of two sequences from two different sets (TF and TFBS).

ID	TF (simplified)	TFBS
1	abbbcdab	TAACG
2	dddbcda	AACGT
3	dabcdbb	CTACA
4	bcdccc	GAACC

Fig. 2. A biological database

Definition 1. ***(Biological database)*** *Let $\Sigma_I = \{I_1, ..., I_m\}$ and $\Sigma_J = \{J_1, ..., J_n\}$ be two sets of m distinct and n distinct letters. Let Σ_I^* and Σ_J^* denote the set of all sequences comprising of letters in Σ_I and Σ_J respectively. A biological database DB is a set of k records $\{R_1, R_2, ..., R_k\}$ between two sequences from the sets Σ_I^* and Σ_J^*. i.e. $R_l = (x_l, y_l)$ where $x_l \in \Sigma_I^*$ and $y_l \in \Sigma_J^*$ for $1 \leq l \leq k$.*

Example 1. A biological database with $\Sigma_I = \{a, b, c, d\}$ and $\Sigma_J = \{A, C, T, G\}$ is shown in Figure 2. Each TF (simplified) is represented as a sequence $x \in \Sigma_I^*$ while each TFBS is represented as a sequence $y \in \Sigma_J^*$. There are $k = 4$ records in the database storing TF-TFBS binding pairs. We will consider the database shown in Figure 2 as an example in explaining the following definitions.

We say that a sequence S is a *subsequence* of another sequence S' if S is a contiguous substring of S'.

Definition 2. ***(Subsequence)*** *A sequence $S = s_1, s_2, ..., s_p$ is a subsequence of $S' = s'_1, s'_2, ..., s'_q$, denoted by $S \preceq S'$, if there exists an integer r where $1 \leq r \leq q - p + 1$ such that $s_1 = s'_r, s_2 = s'_{r+1}, ...,$ and $s_p = s'_{r+p-1}$.*

We say that a record in the database *contains* a sequence if it is a subsequence of the same type of the sequences in the record.

Definition 3. ***(Contain)*** *Suppose there is a record $R = (x, y)$ and two sequences $S \in \Sigma_I^*$ and $S' \in \Sigma_J^*$, S is contained in R if $S \preceq x$ and S' is contained in R if $S' \preceq y$.*

Example 2. The TF sequence "bcd" is contained in all the records in the database.

We adopt similar definitions of support and confidence in traditional association rule mining problem [3].

Definition 4. ***(Support)*** *The support of a sequence S is the portion of records in the database DB that contains S. The support of sequence S and sequence S' is the portion of records in the database that contains both S and S'.*

$$support(S) = \frac{\textit{number of records contain } S}{\textit{total number of records in } DB}$$

$$support(S \cap S') = \frac{\textit{number of records contain } S \textit{ and } S'}{\textit{total number of records in } DB}$$

We say that a sequence is *frequent* in DB if its support is not less than a minimum threshold, denoted by min_sup, defined by the user.

Definition 5. ***(Frequent sequence)*** *Given a user-defined threshold, min_sup, a sequence S is frequent in DB if $support(S) \geq min\_sup$.*

Example 3. Given $min\_sup = 0.8$, the TF sequence "bcd" is a *frequent sequence* because it is contained in all the records in the database and $support(\text{"bcd"}) = 4/4 \geq 0.8$.

An association rule is an implication of the form $X \Leftrightarrow Y$, where $X \in \Sigma_I^*$ and $Y \in \Sigma_J^*$. It means that X and Y are likely to be the binding cores. The degree of likeliness is measured in *forward confidence* and *backward confidence*. Forward confidence ($conf_F$) is the portion of records that contains X in the database that also contains Y. Backward confidence ($conf_B$) is the portion of records that contains Y in the database that also contains X. We say that an association rule has a support s in DB if both X and Y are frequent sequences when $min\_sup = s$.

Definition 6. ***(Association rule)*** *An association rule is an implication of the form $X \Leftrightarrow Y$, where $X \in \Sigma_I^*$ and $Y \in \Sigma_J^*$. The rule has a support s if $support(X) \geq s$ and $support(Y) \geq s$. The forward confidence ($conf_F$) and backward confidence ($conf_B$) of the rule are defined as*

$$conf_F(X \Leftrightarrow Y) = \frac{\textit{number of records contain } X \textit{ and } Y}{\textit{number of records contain } X} = \frac{support(X \cap Y)}{support(X)}$$

$$conf_B(X \Leftrightarrow Y) = \frac{\textit{number of records contain } X \textit{ and } Y}{\textit{number of records contain } Y} = \frac{support(X \cap Y)}{support(Y)}$$

Example 4. "bcd" $\Leftrightarrow$ "AAC" is an association rule with $support = 0.75$, $conf_F = 0.75$ and $conf_B = 1$. It is because $support(\text{"bcd"}) = 1$, $support(\text{"AAC"}) = 0.75$ and $support(\text{"bcd"} \cap \text{"AAC"}) = 0.75$. So,

$$conf_F(\text{"bcd"} \Leftrightarrow \text{"AAC"}) = \frac{0.75}{1} = 0.75$$

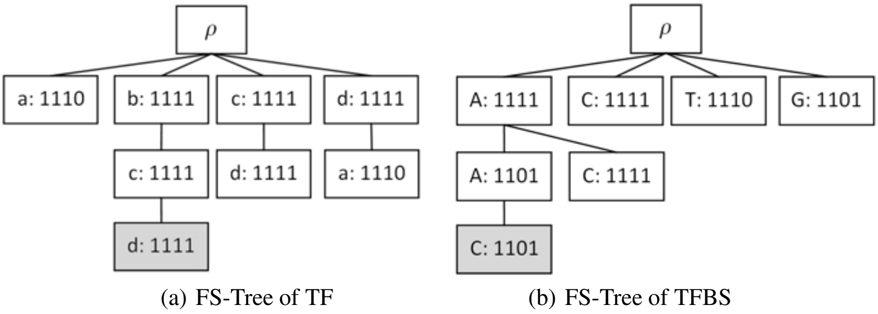
$$conf_B(\text{"bcd"} \Leftrightarrow \text{"AAC"}) = \frac{0.75}{0.75} = 1$$

Problem Definitions

Given a biological database DB, the problem of mining protein-DNA association rules is to find all co-existing subsequences in the two sets of sequences in DB with a minimum support s and both forward and backward confidence c.

4 The Algorithm

Before mining association rules, the algorithm constructs two compact intermediate data structures to facilitate the mining process. The data structures are similar to suffix trees [24], which are widely used in solving string matching problems.



(a) FS-Tree of TF (b) FS-Tree of TFBS

Fig. 3. FS-Trees built from database shown in Figure 2

4.1 Frequent Sequence Tree

We first illustrate the structure of a frequent sequence tree (FS-Tree) by an example. Consider the FS-Tree shown in Figure 3(b), which is constructed from the database shown in Figure 2. It has a null root node, denoted as ρ. A set of prefix sub-trees is the child of it. Each non-root node in the tree stores a letter and a bit vector. Each non-root node represents a sequence formed by concatenating the letters along the path from the root node to itself. The shaded node in 3(b) represents the sequence "AAC" as the letters along the path from the root to the node are "A", "A" and "C". The bit vector in a node indicates whether records in the database contain the representing sequence. The bit vector "1101" stored in the shaded node in Figure 3(b) indicates that the sequence "AAC" is contained in the first, second and last records in the database.

Definition 7. ***(Frequent sequence tree)*** *A frequent sequence tree (FS-Tree) is a tree structure defined as follows:*

1. *It has a null root node denoted as ρ, a set of prefix sub-trees is the child of it.*
2. *Each non-root node stores a letter and a bit-vector.*
3. *Each non-root node represents a sequence by concatenating the letters along the path from the root node to itself.*
4. *The vector in a non-root node indicates whether records in the database contain the representing sequence.*

4.2 Construction of FS-Tree

Based on the definition above, we have two construction algorithms to build a FS-Tree.

Level-by-Level Construction. The first algorithm builds a FS-Tree iteratively with increasing depth. First, it creates the root of the tree. For each letter I in Σ_I, a child with letter I is appended to the root node. Database scans are performed to check whether the records in the database contain the representing sequences. If a sequence is contained in a record, the corresponding bit in the bit vector of the node is set to 1; otherwise, it is set to 0. After updating the bit vector of each child node, the support of the representing sequence is computed and checked if it reaches the minimum threshold. The support can be efficiently computed by counting the number of ones in the bit vector. As the number of ones in a bit vector is the number of records in the database which contain the representing sequence, the support of the representing sequence is the number of ones in the bit vector divided by the total number of records in the database. If the support is less than the threshold, the node is removed and its descendants will not be constructed at next iteration because the representing sequences cannot be frequent.

Lemma 1. *For any two sequences X and X', where $X \preceq X'$ with $r = 1$, if X is not a frequent sequence in DB, X' cannot be a frequent sequence in DB.*

Proof. Suppose X' is a frequent sequence in DB, then more than $(min\_sup \times |DB|)$ of records in DB contain X'. As X is a subsequence of X', the records that contain X' also contain X. Therefore, X is also a frequent sequence, which contradicts to the assumption that X is not a frequent sequence in DB. ■

By skipping those nodes, the size of the structure and the construction time are reduced. Also, irrelevant information is eliminated from the tree. The process repeats iteratively with increasing depth appending child nodes to each node in that level, until the depth of the tree equals to the maximum length of sequences in the database or there is no node in that level.

Multi-level Construction. If the main memory is sufficient, we can build a FS-Tree in a multilevel way to reduce the number of database scans.

Lemma 2. *For any two sequences X and X', where $X' \preceq X$ with $r = 1$, if X is contained in a record R, X' is also contained in R.*

Proof. If X' is not contained in R, any sequence having X' as subsequence cannot be contained in R. Therefore, X cannot be contained in R, which contradicts to the assumption. ■

By using lemma 2, we modify our algorithm as follows. First, we build a full FS-Tree of specified depth k. One database scan is needed to update the bit vectors in all nodes in the tree by checking the leaf nodes. Then, we apply tree pruning in a top-down manner, deleting the nodes with supports less than the threshold.

Example. Consider the database shown in Figure 2, and let $min\_sup = 0.75$, the FS-trees constructed are shown in Figure 3.

4.3 Generating Association Rules

Once we have the two TF-Trees, the process of generating association rules is relatively simple. For each node on one of the FS-Tree, we use a depth-first traversal on the other FS-Tree to generate candidate association rules. Then, the forward and backward confidences of the rules are calculated. The confidences of a candidate rule $X \Leftrightarrow Y$ can be efficiently computed by first bitwise intersecting the bit vectors of the two nodes.

Table 1. Biological data used in experiments

	Number of sequences	Min length	Max length	Avg length
TF	708	70	2717	494
TFBS	15668 (grouped into 708 records)	4	269	22

The forward confidence of the rule equals to the number of ones in the resultant bit vector divided by the support of X. The backward confidence of the rule equals to the number of ones in the resultant bit vector divided by the support of Y. Those candidate rules with confidences reaching the threshold are output by the algorithm.

For example, consider the rule “bcd” $\Leftrightarrow$ “AAC” generated form the shaded nodes in the FS-Trees in Figure 3, we intersect the bit vectors of the two nodes to get the resultant bit vector “1101”. The forward confidence of this rule is the number of ones in the resultant vector divided by $support$(“bcd”), which is $3/4 = 0.75$. Similarly, the backward confidence of this rule is $3/3 = 1$.

In order to prevent enumerating all the possible pairs in the traversal, we reduce the search space by two means.

Pruning Infrequent Sequences. As the FS-Trees only contain nodes of frequent sequences, computation time is reduced as unqualified rules having infrequent sequences on either side of the rules are not generated and their confidences are not computed.

Pruning Rules with Low Confidences. If we find that the forward confidence of the generated rule $X \Leftrightarrow Y$ is less than the minimum threshold, we can skip traversing all the descendants of the node representing Y as the rules generated from its descendants cannot reach the minimum threshold.

Lemma 3. *For any sequence X and any sequence Y and Y' such that $Y \preceq Y'$ with $r = 1$, if $X \Leftrightarrow Y$ is not a qualified association rule (with a low forward confidence), $X \Leftrightarrow Y'$ cannot be a qualified association rule.*

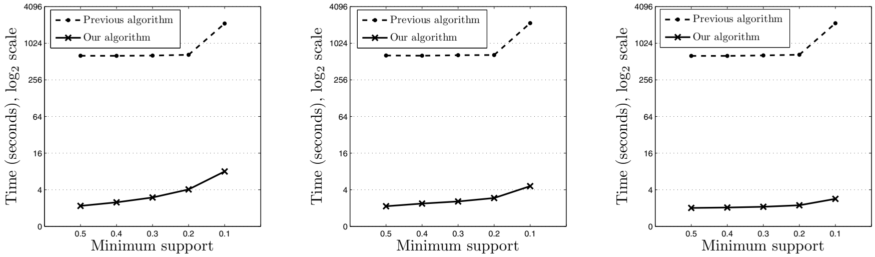
Proof. If $X \Leftrightarrow Y$ is an unqualified association rule with its forward confidence less than the threshold min_conf.

$$\begin{aligned} conf_F(X \Leftrightarrow Y) &\leq min\_conf \\ support(X \cap Y) &\leq min\_conf \times (support(X)) \\ support(X \cap Y') \leq support(X \cap Y) &\leq min\_conf \times (support(X)) \\ conf_F(X \Leftrightarrow Y') &\leq min\_conf \end{aligned}$$

Therefore, $X \Leftrightarrow Y'$ is not a qualified association rule. ■

5 Experimental Result

We evaluated the proposed algorithm using multilevel construction on real biological data extracted from TRANSFAC (Professional ver 2009.4). TFBS data were grouped



(a) min_conf = 0.5 (b) min_conf = 0.7 (c) min_conf = 0.9

Fig. 4. Performance comparison in term of CPU Time

by TF because the same TF can bind to more than one TFBS. In our experiments, we consider a group of TFBSs as a single record. The statistics of the data are given in Table 1. All the experiments were conducted on a PC with 2.93GHz Dual Core CPU and 8GB memory running Ubuntu Desktop.

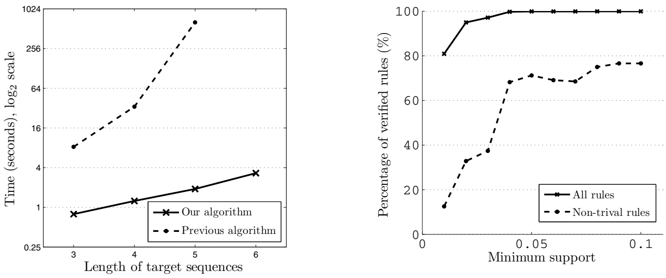
5.1 Comparative Performance

We compare our algorithm with that in [15] and focus on efficiency and scalability since both model the associated TF-TFBS binding core discovery problem with exact subsequences. As the previous algorithm can only handle fixed lengths of subsequences, we constrain our algorithm accordingly to construct FS-Trees with the maximum depth. First we compare their performance as we increase the length of subsequences (length of binding cores). Figure 5(a) shows how the algorithms performed when the length of subsequences (maximum depth of FS-Trees) was increased from 3 to 6. A minimum support of 0.7 and a minimum confidence of 0.5 were used.

The second set of experiments compared the performance of the algorithms when the support and confidence varied. The maximum lengths of the subsequences were limited to 5 in these experiments. Figure 4(a), 4(b) and 4(c) show how the algorithms performed when minimum confidences of 0.5, 0.7 and 0.9 were used respectively. We cannot compare the performances of the algorithms when the minimum support is less than 0.1 as the previous algorithm took more than ten hours to run while our algorithm took less than a minute.

Experimental results show that our algorithm is several orders of magnitude faster than the previous algorithm. The reason is that the previous algorithm wastes much time in generating useless itemsets and rules. As the alphabet sizes of the sequences are short (4 for TFBS and 20 for TF), short sequences are frequently contained in each record. Therefore the size of large 1-itemsets and large 2-itemsets in the *Apriori* algorithm is enormous. Moreover, many candidates generated from those large itemsets are useless as they contain only one type of sequences (either TFs or TFBSs on both sides). Furthermore, the use of FS-Trees provides efficient methods for calculating supports of sequences and confidences of rules.

For space complexity, the current version is costly to some degree but it is efficient and scalable for the real application on TRANSFAC, where the annotated data grows



(a) CPU Time against maximum depth

(b) Percentage of verified rules against minimum support

Fig. 5. Experimental Results

sub-linearly. Moreover, like frequent patterns mining problem, our approach suffers from big redundancy in the results. However, the algorithm supports simple compression using approximation. We will further address these two issues in our future work.

Although the experiments in this paper are for exact associated binding cores based on [15], the efficiency and scalability of FS-Trees are demonstrated in detail. The FS-Trees show much better applicability to further generalizations such as approximate binding core identification in our recent extension.

5.2 Applicability to Predicting Binding Cores

In order to show that our algorithm is applicable to predicting protein-DNA binding cores, we verified the generated association rules with a biological database named Protein Data Bank (PDB). PDB stores 3D structures of protein-DNA interactions captured by expensive experiments. For each 3D structure of a TF-TFBS interaction, we extracted the fragments of a TF and a TFBS such that the distances between them are within 3.5Å, meaning that they are bound together. We compared the generated association rules with these fragments. A rule $X \Leftrightarrow Y$ is said to be verified if both X and Y contains in the TF and TFBS respectively in at least one of the fragments. More precisely, we removed those trivial rules which contains sequences of length less than 2 in either side and computed the percentage of verified non-trivial rules. In our previous study [15] we have shown that randomly generated rules (of lengths = 5) are unlikely to be verified by PDB (verified percentage $\ll 10\%$). It is the indirect evidence that the verified association between TF and TFBS subsequences is not likely to happen purely by chance.

Figure 5(b) shows the percentages of rules that can be verified by PDB with 0.5 as the minimum confidence. The verification result shows that nearly all association rules with high minimum support and confidence can be verified. Although certain amount of trivial rules (length ≤ 2) shall be removed, the verified percentage is still up to 76.6%. It is a normal phenomenon as the alphabet sizes of TF and TFBS are small, sequences of short length usually exists in every binding cores. Moreover, PDB does not cover all the possible binding pairs and we believe that those unverified rules are those undiscovered binding cores which scientists can perform experiments to verify.

In comparison with the results obtained in [15], which has no confidence control and is limited by the maximal length 6, results generated by our algorithm cover almost all of their results (covering 96.5% and 100% for 5-5 (TF length-TFBS length) and 6-6 rules in [15] by lowing the confidence threshold to 0.1). Furthermore, on lengths $\geq$ 6, we have 133 verified rules, compared with only 6 in [15] (limited by the maximal length). More about the correctness of the rules and PDB verification will be addressed in an extended version.

6 Conclusion

In this paper, we have formally introduced the problem of mining protein-DNA association rules over biological databases. We also proposed an efficient algorithm using compact structures called FS-Trees. Experiments on real biological data demonstrated significant speed up over our previous work. Verification on the generated rules confirmed the applicability of the proposed methods on predicting protein-DNA binding cores. Although our algorithm is initiated by protein-DNA interactions, our framework is general enough for other similar problems such as protein-protein interactions. Furthermore, the FS-Trees serve as a useful basis for more general extensions we are working towards recently, e.g. approximate associated binding core discovery [9].

References

1. Savasere, A., Omiecinski, E., Navathe, S.: An efficient algorithm for mining association rules in large databases. In: Proc. 1995 Int. Conf. Very Large Data Bases, pp. 432–443 (1995)
2. Agarwal, R.C., Aggarwal, C.C., Prasad, V.V.V.: A Tree Projection Algorithm for Generation of Frequent Item Sets. Journal of Parallel and Distributed Computing 61(3), 350–371 (2001)
3. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. ACM SIGMOD Record 22, 207–216 (1993)
4. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc. 20th Int. Conf. Very Large Data Bases, VLDB, vol. 1215, pp. 487–499. Citeseer (1994)
5. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Proceedings of the Eleventh International Conference on Data Engineering, pp. 3–14. IEEE (1995)
6. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 429–435. ACM, New York (2002)
7. Bailey, T.L., Elkan, C.: Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In: Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology, pp. 28–36 (1994)
8. Brin, S., Motwani, R., Ullman, J.D., Tsur, S.: Dynamic itemset counting and implication rules for market basket data. SIGMOD Rec. 26, 255–264 (1997)
9. Chan, T.M., Wong, K.C., Lee, K.H., Wong, M.H., Lau, C.K., Tsui, S.K., Leung, K.S.: Discovering approximate associated sequence patterns for protein DNA interactions. Bioinformatics 27(4), 471–478 (2011)
10. Das, A., Ng, W.K., Woon, Y.K.: Rapid association rule mining. In: Proceedings of the Tenth International Conference on Information and Knowledge Management, CIKM 2001, pp. 474–481. ACM, New York (2001)

11. Galas, D.J., Schmitz, A.: Dnaase footprinting a simple method for the detection of protein-dna binding specificity. Nucleic Acids Research 5(9), 3157–3170 (1978)
12. Garner, M.M., Revzin, A.: A gel electrophoresis method for quantifying the binding of proteins to specific dna regions: application to components of the escherichia coli lactose operon regulatory system. Nucleic Acids Research 9(13), 3047–3060 (1981)
13. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. ACM SIGMOD Record 29(2), 1–12 (2000)
14. Jones, S., van Heyningen, P., Berman, H.M., Thornton, J.M.: Protein-DNA interactions: a structural analysis. Journal of Molecular Biology 287(5), 877–896 (1999)
15. Leung, K.S., Wong, K.C., Chan, T.M., Wong, M.H., Lee, K.H., Lau, C.K., Tsui, S.K.W.: Discovering protein-DNA binding sequence patterns using association rule mining. Nucleic Acids Research 38(19), 6324–6337 (2010)
16. Li, M., Ma, B., Wang, L.: Finding similar regions in many sequences. Journal of Computer and System Sciences 65, 73–96 (2002)
17. MacIsaac, K.D., Fraenkel, E.: Practical strategies for discovering regulatory DNA sequence motifs. PLoS Comput. Biol. 2(4), e36 (2006)
18. MacIsaac, K.D., Fraenkel, E.: Practical strategies for discovering regulatory dna sequence motifs (2006)
19. Matys, V., Kel-Margoulis, O.V., Fricke, E., Liebich, I., Land, S., Barre-Dirrie, A., Reuter, I., Chekmenev, D., Krull, M., Hornischer, K., Voss, N., Stegmaier, P., Lewicki-Potapov, B., Saxel, H., Kel, A.E., Wingender, E.: Transfac and its module transcompel: transcriptional gene regulation in eukaryotes. Nucleic Acids Research 34, 108–110 (2006)
20. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New algorithms for fast discovery of association rules. In: 3rd Intl. Conf. on Knowledge Discovery and Data Mining, vol. 20, pp. 283–286 (1997)
21. Park, J., Chen, M., Yu, P.: An effective hash-based algorithm for mining association rules. ACM SIGMOD Record 24(2), 175–186 (1995)
22. Pavesi, G., Mauri, G., Pesole, G.: An algorithm for finding signals of unknown length in DNA sequences. Bioinformatics 17, S207–S214 (2001)
23. Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: ICCCN, p. 215. IEEE Computer Society (2001)
24. Sagot, M.-F.: Spelling Approximate Repeated or Common Motifs using a Suffix Tree. In: Lucchesi, C.L., Moura, A.V. (eds.) LATIN 1998. LNCS, vol. 1380, pp. 374–390. Springer, Heidelberg (1998)
25. Smith, A.D., Sumazin, P., Das, D., Zhang, M.Q.: Mining chip-chip data for transcription factor and cofactor binding sites. Bioinformatics 21(suppl.1), i403–i412 (2005)
26. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: Advances in Database Technology XEDBT 1996, pp. 1–17 (1996)
27. Wang, K., Tang, L., Han, J., Liu, J.: Top Down FP-Growth for Association Rule Mining. In: Chen, M.-S., Yu, P.S., Liu, B. (eds.) PAKDD 2002. LNCS (LNAI), vol. 2336, pp. 334–340. Springer, Heidelberg (2002)
28. Wang, K., Xu, Y., Yu, J.: Scalable sequential pattern mining for biological sequences. In: Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management, pp. 178–187. ACM, New York (2004)
29. Zaki, M.: Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering 12(3), 372–390 (2000)
30. Zaki, M.: SPADE: An efficient algorithm for mining frequent sequences. In: Machine Learning, pp. 375–386 (2001)

A Novel Approach for Finding Alternative Clusterings Using Feature Selection

Vinh Thanh Tao and JongHyeok Lee

Pohang University of Science and Technology
{vinhtt,jhlee}@postech.ac.kr

Abstract. Alternative clustering algorithms target finding alternative groupings of a dataset, on which traditional clustering algorithms can find only one even though many alternatives could exist. In this research, we propose a method for finding alternative clusterings of a dataset based on feature selection. Using the finding that each clustering has a set of so-called important features, we find the possible important features for the altenative clustering in subsets of data; we transform the data by weighting these features so that the original clustering will not likely to be found in the new data space. We then use the incremental K-means algorithm to directly maximizes the quality of the new clustering found in the new data space. We compare our approach with some previous works on a collection of machine learning datasets and another collection of documents. Our approach was the most stable one as it resulted in different and high quality clusterings in all of the tests. The results showed that by using feature selection, we can improve the dissimilarity between clusterings, and by directly maximizing the clustering quality, we can also achieve better clustering quality than the other approaches.

Keywords: data clustering, k-means, clustering quality, clustering dissimilarity.

1 Introduction

Data clustering's objective is to group similar data, in terms of some data attributes (features), into the same class and dissimilar data into different classes to provide a general view about the data [5]. Most of the clustering algorithms can only find one *way to group the data* (*clustering*) even though there would be some alternatives. It means most of them provide only one view regardless of the other views about the data. Alternative clustering algorithms address this problem as they try to find valuable and different clusterings of a dataset.

The most typical example for the need of alternative clustering is described in Fig. 1 where both clusterings C_o and C_a are valuable and different from each other. The clusters in C_o are formed based on data points similar in terms of feature y, while the clusters in C_a are groups of data with similar value of feature x; in other words, y is the important feature for the clustering C_o and x is the

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 482–493, 2012.
© Springer-Verlag Berlin Heidelberg 2012

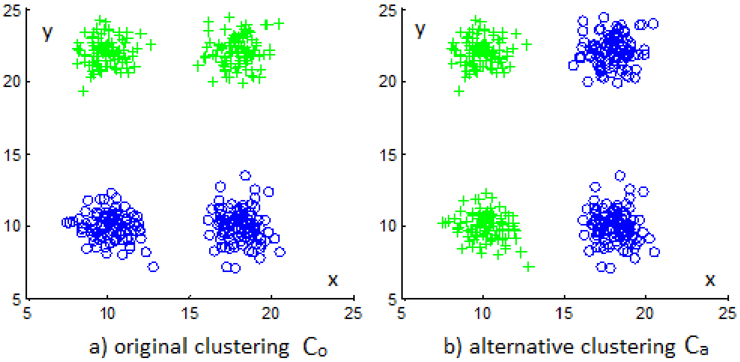


Fig. 1. Original and alternative clusterings of our synthetic dataset. Green crosses and blue circles represent data points in different clusters.

important feature for the clustering C_a. In the general case, more complex data may have different clusterings, and each of them are of high quality as well as different from the others. This is especially correct for high dimensional data since it can be classified based on a number of different sets of data attributes.

Alternative clustering algorithms address those problems as they aim to find multiple ways of grouping data, each of which is high quality and different from the others, with much lower cost and run times. They condition on the original clustering to find the alternative so that the alternative will be different than the origin, meanwhile, they maximize the quality of the new clustering. The dissimilarity is measured by the ratio of common data pairs between clusterings. The quality is measure by a metric quantifying how data in the same cluster close to each other and data in the different clusters far from each other. They are given in details in the Chapter 3. In the most general case, the alternative clustering problem is defined as follows.

Problem 1. *Semi-supervised: given an initial clustering C_o of the data X, find another clustering C_a of X that is high quality and different from, C_o.*

Problem 2. *Unsupervised: find two clusterings C_o and C_a of X that are high quality and different from each other.*

The unsupervised approach is usually a sequential combination of a data clustering algorithm and another Semi-supervised alternative clustering algorithm, thus we only focused on the Semi-supervised approach in this research. To sum up, alternative clustering aims to classify data into more than one clusterings; the fundamental objectives for this problem are clusterings‘ quality and dissimilarity.

There has been a very limited number of approaches to this problem proposed. We generally classified them into two main groups distinguishing by their behavior, namely, data transformation based and non-data transformation based. Most of the approaches in the first group [7, 4, 3] made use of some concepts from information theory such as mutual information to define the difference between clusterings. Mutual information is a measurement of similarity between the distributions of two datasets. By minimizing this value, they expect the dissimilarity will also be minimized. These approaches have the same characteristic is that, they maximize the quality and the dissimilarity based on the other criteria rather than the distance metric used in the quality and dissimilarity measurements. They also do not make use of the data features, fundamental factor for distinguishing data objects, thus leaving much room for other methods. The general idea of the second group, including [10] and ours, is to transform the data into a new space, in which the original clustering is likely to disappear and a different clustering is easily found using any clustering algorithm. It has been proved by the results of [10] and ours that we have better dissimilarity than the former methods. Compare to our approach, Qi and Davidson's with their assumptions about the distribution of data derives in lower quality clusterings (Table 1). Their use of the distributions of clusterings instead of maximizing the metric quality could also be accounted for the low quality.

In this paper, we presented another approach to the alternative clustering problem; our key idea is to find alternative clusterings using feature selection and directly maximize the quality of data clusterings. We ourperform most of the previous approaches by using data transformation method. We directly maximize the clustering quality based on metric distance of data objects rather than using other criteria. According to our classification, our algorithm is the second one using data projection approach in this line, however, this is the first one to maximize clustering quality using metric distances between data objects. The experiment results showed that the approaches using data transformation, including ours and [10], had better dissimilarity than the others (Table 2). Moreover, our approach always results in higer quality clusterings than those of [10] (Table 1). These results support our claims that approaches using feature selection result in better clusterings' dissimilarity and that directly maximizing clustering quality derives in better quality clusterings.

2 Previous Works

Since alternative clustering is very new to researchers, there have been limited works on this field of data mining. Approaches to alternative clustering could be categorized into two main groups, one is non-data transformation based, the other is data transformation based. This classification is made based on the behavior of using data of the approaches.

2.1 Non-data Transformation Based Approaches

Approaches in the first group used some information theory concepts, including entropy and mutual information, to ensure the quality and the dissimilarity of the clusterings. Conditional information bottleneck (CIB) [7] was the very first representative of all; it implemented a semi-supervised method in which an alternative clustering is found using the existing clustering as negative information. CIB maximized the shared information between the cluster labels and data features to ensure the clustering is of high quality, while conditioning on the given clustering for a low similarity. The main drawback of this approach is that CIB must have joint distribution information for each variable which may not be available for all the data. **CAMI** (Clustering for Alternatives with Mutual Information) [3], as an unsupervised approach, further improved the idea by removing the joint distribution information. It made use of the mixture multivariate Gaussian distribution of the data for finding two high quality clusterings simultaneously without any prior-knowledge about the original data classification. CAMI, as a matter of fact for approaches in this group, minimized the mutual information between clusterings for their dissimilarity. It overcame the difficulty of the joint distribution with CIB, however, the assumption of the mixture model of multivariate Gaussian distribution of the data made it unsuitable for data of non-Gaussian shape. The recent **NACI** (Non-linear Alternative Clustering with Information theory) [4] targeted the alternative clustering problem only using the information theory concepts as indicated in its name. NACI used the same approach with CIB in which the mutual information between cluster labels and data features is maximized for the quality, while the mutual information between clusterings is minimized for their dissimilarity. However, NACI motivated the objectives of their method differently through the use of an information theory concept named Fano's inequality. The advantage of NACI to previous works in this group is that, without any assumption about data distribution, this method works well with arbitrary shapes of data of any scale.

The most typical approach for the data constraint based group is **COALA** (Constrained Orthogonal Average Link Algorithm) [1]. COALA is a semi-supervised method which exploits the effectiveness of pairwise constraints; it generates a set of cannot-link constraints from the provided clustering and feeds them to an agglomerative hierarchical clustering algorithm. This process is to ensure that given two clusters in the new clustering, they cannot be merged by the clustering algorithm if they contain any pair of objects in the provided clustering; or in other words, this process is to ensure the dissimilarity. COALA also made use of two kinds of merge each of which is performed in different cases to manipulate the tradeoff between the quality and the dissimilarity of the alternative clustering. The problem of this approach is the complexity of the algorithm; this complexity makes it not applicable for large scale data.

2.2 Data Transformation Based Approaches

The general idea of these approaches is to transform the data into a new space in which the original clustering is likely to disappear and a different clustering is easily found using any clustering algorithm. Qi and Davidson [10] try to transform the data into a new space so that data objects are more likely to be assigned to a cluster other than the projection of their old cluster in the new space; this process was to make the two clusterings different; in practice, they minimized the probability density function of the projected data. They exploited another information theory concept, named Kullback-Leibler divergence, to guarantee the clustering quality; they minimized this divergence to reserve the property of the old data in the new space so that the quality of the clustering in the new space is ensured.

Our approach shares the same basic idea with [10], however, we used a different method to project the data based on data features: we find on data subsets the possible important features for alternative clusterings; we transformed the data by weighting these features against the important features of the original clustering. Our approach outperformed the others in most of the experiments.

3 Materials and Methods

It is proved that different features affect clusters differently and similarity between a pair of data points is due to different features [5]. Based on these previous results, we found that clusterings differ from each other by a set of features – the so called important features for the clustering. Our approach is motivated by this factual information through the choice of data transformation approach and metric based clustering evaluation.

3.1 Measurements

According to their objective of alternative clustering algorithms, clusterings are evaluated by their quality and the dissimilarity between them. These quantities are measured using Dunn Index (DI) [2] and Jaccard Index (JI) [9], respectively.

Dunn Index measures the ratio of the minimum distance between two clusters to the maximum cluster diameter: Let $C = \{c_1..c_k\}$ be a clustering where c_j is the centroid of a cluster of C, $\delta : C \times C \rightarrow \mathbb{R}^+$ be a cluster-to-cluster distance and $\Delta : C \rightarrow \mathbb{R}_0^+$ be a cluster diameter measure, then DI is

$$DI(C) = \frac{min_{i \neq j}\delta\{(c_i, c_j)\}}{max_{1 \leq l \leq k}\{\Delta(c_l)\}} \tag{1}$$

The quality of a clustering increases with its DI. Dunn Index has been proven to be an effective measure compared to the others in measuring clustering quality [1] and is used in all of the previous approaches. Our approach is motivated by the use of some metric distance to calculate the similaritiy between data objects, so the use of a metric based distance is more obvious the use of a data distribution based distance.

Jaccard Index measures the ratio of common data object pairs between clusterings. A pair is two data object in the same cluster. The Jaccard Index is defined as follows:

$$JI(C, S) = \frac{|C| \cap |S|}{|C| \cup |S|}$$

where $|C|$ and $|S|$ are the numbers of pairs in C and S, respectively. The dissimilarity of two clustering decreases when their JI increases.

3.2 Data Transformation and Clusterings' Dissimilarity

The matrix inversion in [10] will put more weight on unimportant features than the important features of the original clustering. This is equal to inversing the selection of features and does not always work well. In our approach, we chose a more natural and straight way to select features. We only select those based on which the data can really be classified. This is described in the followings.

Our Approach. We propose to use the data transformation approach for its advantages against the non-data transformation ones. The processes of our method are quite straight:

Feature Selection: another combination of data features must be selected so that, the original clustering will not likely to be found on the transformed data. In practice, we selected those important for further classifying each data cluster.

Data Transformation: the data is transformed based on the selected features. We chose to weight the features to make them more important than those original for the original clustering.

Finding Clustering: on the new data, we apply our clustering algorithm whose objective function uses the same criteria to those for the quality measurement in order to derive in best clustering quality.

The detailts of those processs is described in the next sections.

Important Features. This is the fundamental concept for the data transformation approach. In our research, we defined a set of features is called important if they contribute the most to the *formation* of a clustering. The formation of a clustering as the distances between the centroids of that clustering. For example, the formation of the clustering in Fig. 1a is the distance between two centroids; it is mostly contributed by the feature y rather than the feature x, thus y is the important feature for the formation of the clustering. In practice, where the number of features is very large, we will select a set of features whose total contribution is more than a certain threshold.

Features are selected as important in the descending order of their importance; the selection procedure is stopped when it meets a threshold; this threshold is a certain ratio of the sum of important features and that of all features. This selection procedure is described in the Algorithm 1. In our experiments, the threshold is set to 0.9 to select the major of features.

Possible Important Feature Selection. The matrix inversion in [10] virtually flips the feature sets whereas the set of important feaures becomes unimportant and the set of unimportant features becomes important. This is not a good feature selection since a good combination of features may have both previously important and unimportant features.

In our approach, we only select sets of features that are really able to separate the data. Our solution is to select a subset of data where the original important features seem not to be important for subclustering (finding a clustering of)

Algorithm 1. Important Feature Selection

```
$total\_importance \leftarrow 0$
$total\_distance \leftarrow \Sigma_i(feature\_distances(i))$
$sort\_in\_descending\_order(feature\_distances)$
for $i = 1$ to $k$ do
  $total\_importance \leftarrow total\_important + feature\_distances(i)$
  if $total\_importance/total\_distance \geq threshold$ then
    break
  end if
end for
```

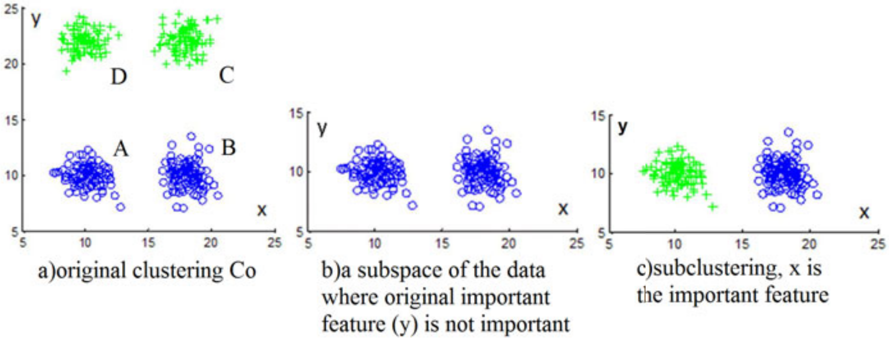


Fig. 2. Illustration of finding possible important features in subspaces

this subset. We regard these features as the possible important features for the alternative clustering for the complete data.

Our feature selection method is well illustrated in (Fig. 2). A, B, C, and D are four Gaussian sub-clusters of the original clustering of $C_o = \{(AB), (CD)\}$ of our synthetic dataset (Fig. 2a); the important feature for C_o is y. By further clustering the cluster (AB) of C_o into sub-clusters A and B, we found x is the important for this sub-clustering.

Data Transformation. The transformation is performed by weighting the possibile important features so that, the contribution of those possible important will be more than that of those original important to the original clustering. By doing so, the orginal clustering will not likely to be found. The process of transforming is given in Algorithm 2 and Fig. 3, where β is set to 2 in our experiments and the set of important features for the original clustering and the alternative clustering are F_o and F_a, respectively.

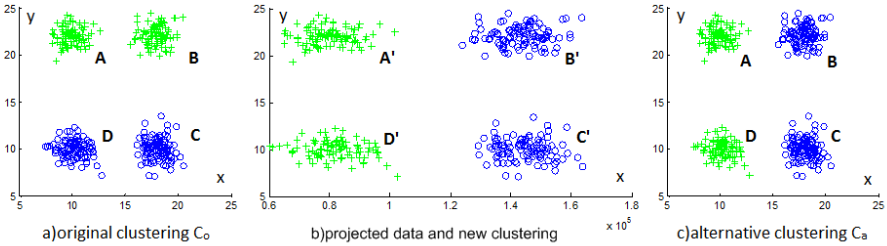


Fig. 3. Illustration of how data is transformed and how alternative clustering found on transformed data

Algorithm 2. Weighting Features

```
$dist_1 \leftarrow \Sigma_{i \in F_o} contribution(i)/|F_o|$
$dist_2 \leftarrow \Sigma_{j \in F_a} contribution(j)/|F_a|$
$\alpha \leftarrow \beta * dist_2/dist_1$
$x_{ij} = x_{ij} * \alpha, \forall i \in [1..n], \forall j \in F_a$
```

Finding Alternative Clustering. This section describe how we find and maximize the quality of the alternative clustering on the transformed data.

Objective of Clustering Algorithm: Recall the use of a metric based clustering algorithm to directly maximize the quality of the clustering (Equation 1). Directly maximizing DI equals to directly maximzing the inter-cluster distance and/or minimizing the intra-cluster distance. Given the cluster c_i and cluster c_j as those whose inter-cluster distance is the minimum of the clustering. The movement of a data object x of c_i to c_j will be decided based on the impovement of clustering quality it made when moving x. This quality change is given as below whereas $c_{i'}$ and $c_{j'}$ are the clusters c_i and c_j respectively after moving the data object x. In the case of moving x also affects the maximum intra-cluster distance, the quality change will be the maximum of changes made by moving x to another cluster.

$$\max \mathit{\Delta} DI(C) = \arg\max_j \frac{\delta\{(c_i, c_j)\} - \delta\{(c_{i'}, c_{j'})\}}{max(\mathit{\Delta}(c_{i'}), \mathit{\Delta}(c_{j'}))} \tag{2}$$

The denominator in the above equation is in fact identical to the objective of the K-means clustering algorithm since K-means minimizes the sum square error. For this reason, we used K-means clustering algorithm with incremental steps as our clustering algorithm for directly maximizing the clustering quality.

Batch K-means Clustering Algorithm: randomly selects k initial centroids; it reassigns the data points to the nearest cluster; it recalculate the new centroid for each cluster; this process iterates until the criterion function converges [8]. This classical approach has two major drawbacks [6]: 1) The clustering quality significantly depends on the choice of the initial partition and 2) The algorithm

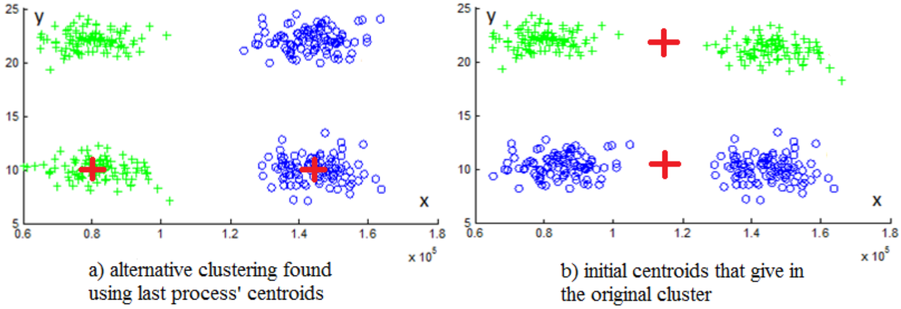


Fig. 4. Illustration of choosing different starting centroids may derive in different clusterings

may easily fall into the local minimum trap even for very simple datasets. *K-means clustering algorithm with Incremental steps* is used to addresses these problems. It makes an addition to the reassignment of a data point x from π_i to π_j $(i \neq j)$ based on the sign of

$$\Delta = \sum_{i \in \pi_i} ||x_i - c_i||^2 + \sum_{i \in \pi_j} ||x_i - c_j||^2 - \sum_{i \in \pi_i - o} ||x_i - c_{i'}||^2 - \sum_{i \in \pi_j + o} ||x_i - c_{j'}||^2 \quad (3)$$

where Δ is the change of the partition‘s quality it made when moving a data point from π_i to π_j, $c_{i'}$ and $c_{j'}$ are the new centroids of π_i and π_j respectively.

Initial centroid selection: Even for the K-means with incremental steps, the algorithm still easily falls into the same local trap. This is well illustrated through the Fig. 4b where even with the transformed data, the original clustering can still be found by selecting proper initial centroids. The best solution for this problem is to carefully choose the starting centroids for the algorithm. We use the centroids from the previous process as starting centroids for finding the alternative clustering (Fig. 4a).

4 Results

Since the DI scores depend on the type of distance measurement used in each approach, we only compared our DI score to that of [10]; we used GDI33 [2], a variation of DI. We ran each approach ten times for all the databases and took the average results; results of these approaches are then compared with those reported in [3] [1] [4] and [7]. The implementation of [10] was provided by its authors. We used our synthetic data to test our algorithm and others databases from UCI[1], namely, Segmentation, Vehicle, Vowel, Ionosphere, and Glass for evaluating the performance compared to the others.

[1] http://archive.ics.uci.edu/ml/

Table 1. Alternative clustering quality and dissimilarity results of our approach compared to [10]'s. Higher DI is better quality; lower JI is better dissimilarity.

Dataset	Segmentation		Vehicle		Vowel		Ionosphere		Glass	
Measurement	DI	JI	DI	JI	DI	JI	DI	JI	DI	JI
Original	0.53	1.00	0.56	1.00	0.46	1.00	0.65	1.00	0.21	1.00
[10]	0.14	0.30	0.72	**0.21**	**0.74**	**0.12**	0.77	0.46	0.54	**0.31**
our approach	**0.86**	**0.24**	**1.39**	0.25	**0.74**	**0.12**	**0.79**	**0.42**	**0.81**	0.36

4.1 Synthetic Datasets

The dataset consists of 4 Gaussian sub-classes each of which was made up from 100 two-dimensional data points; these sub-classes were originally classified into 2 classes (Fig. 1a). This is the simplest dataset for testing the ability of finding alternative clustering used in most approaches [3] [1] [10] [4].

We tested this dataset with our approach and [10], then we compared the dissimilarities to the published results of COALA, CAMI, NACI, and CIB. Since DI depends on the distance measurement, we only compared our DI with that of [10] and the original clustering of each dataset.

The experimental results (Table 1) and those reported in [3] [1] [4] and [7] (Table 2) showed that, except CIB and CAMI, most of the approaches could find the desired alternative clustering (Fig. 1b). Our implementation worked well and derived in the best results.

4.2 UCI Datasets

We tested the implementations of our approach and [10] on five UCI datasets, including Segmentation, Vehicle, Vowel, Ionosphere, and Glass; then we compared our results of dissimilarity with those reported of COALA, CAMI, NACI, and CIB.

In the comparison with the approach in [10], we had significantly better DI and equally good JI scores in all of the experiments (Table 1). We significantly outperformed [10] in the task of clustering quality for four out of five datasets. In the experiment with Segmentation, [10] resulted in 0.14 of DI score while both of our approaches were more than 6 times better than that when we achieved the DI score of 0.86. We almost doubled that value of [10] in the test with Vowel, and were slightly better compared to [10] in the rest. This low quality of [10] is accounted for their not direct maximization of clustering quality.

In the comparison of our approaches and [10]'s to COALA, CAMI, NACI, and CIB, all of the approaches had very competitive JI scores (Table 2), however, our transformation based approaches are the most stable ones as we resulted in best performance for 2 out of 4 tests and always very close to the best approach in the other experiments.

Table 2. Comparison of the dissimilarity of all approaches. The lower, the better.

Dataset	Segment	Vehicle	Vowel	Synthetic
COALA	0.29	0.26	0.27	**0.33**
CAMI	0.27	0.32	**0.11**	0.38
NACI	0.25	0.28	**0.11**	**0.33**
CIB	0.32	0.41	0.26	0.40
[10]	0.30	**0.21**	0.12	**0.33**
our approach	**0.24**	0.25	0.12	**0.33**

Table 3. Experiment with the textual information to see how similar the alternative clustering is to the target clustering

Comparison between	JI
Original and Test	0.062
Alternative and Test	0.291

4.3 Textual Data

In this experiment, we tested our algorithm with a collection of articles from The New York Times to see how effective and scalable it is to very high dimensional textual data. The collection is made up by 3076 articles from the online version of The New York Times[2]. These articles initially grouped into 30 news categories including *business*, *sports*, *US*; they are contributed by 1539 authors.

In the preprocessing, we refined the data to make a Vector Space Model (VSM) for the data. We stemmed the terms and used the normalized *tf.idf* to weight these terms. The result is a VSM of 3076 articles of 41062 terms. To fit the model to the original clustering of the data, reduce the dimension of the model to the important features of the original clustering. The test class label is made up of the classes of authors. Those who contributed less than 10 articles will be grouped together in a class. The result for this process is a collection of 24 authors, each of which accounts for the average of 154 articles.

We run our experiment on this dataset to find out if we can group the articles into the classes of authors using alternative clustering. The resutls are given in the table 3. It's noticable that the alternative clustering and the target clustering got the JI score of 0.291, whereas the JI score for the orginal clustering (by categories) and the target clustering (by authors) is only 0.062. This result illustrated well the capability that we can apply our algorithm to textual data.

5 Discussions

Our approach successfully passed the test with the synthetic and UCI datasets with the most stable performance. The results illustrate well the importance of feature selection to find alternative clusterings. They also prove the significance of the direct maximization of clustering quality.

[2] http://www.nytimes.com/

The results in the experiment with the synthetic dataset has proven that the initial centroid was also important in helping the centroid-based clustering algorithms derive in different clusterings.

6 Conclusion

Our study claimed the importance of feature selection and direct maximization of clustering quality in the problem of finding alternative clusterings. It also proved that initial centroid selection for centroid-based clustering algorithms is significant for the problem.

Data objects differ from the other in terms of some so-called important features. Approaches come with the feature selection will result in better dissimilarity alternative clusterings than the others using mutual information. The results also depend on the strategy of the feature selection. Based on the use of measurement for clustering quality, the approaches directly maximize the quality (maximizing the same criteria with the quality measure) derive in better clustering quality than the other approaches not directly maximize the quality.

The importance of features in clustering will make them useful for exploiting high dimensional data such as textual data and social networks data.

Acknowledgments. This work was supported in part by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korean government (MEST No. 2011-0003029), and in part by the BK 21 Project in 2011.

References

[1] Bae, E., Bailey, J.: Coala: A novel approach for the extraction of an alternate clustering of high quality and high dissimilarity. In: ICDM, pp. 53–62 (2006)

[2] Bezdek, J.C., Li, W.Q., Attikiouzel, Y., Windham, M.: A geometric approach to cluster validity for normal mixtures. Soft Computing - A Fusion of Foundations, Methodologies and Applications 1(4), 166–179 (1997)

[3] Dang, X.H., Bailey, J.: Generation of alternative clusterings using the cami approach. In: SDM, pp. 118–129 (2010)

[4] Dang, X.H., Bailey, J.: A hierarchical information theoretic technique for the discovery of non linear alternative clusterings. In: KDD, pp. 573–582. ACM (2010)

[5] Dash, M., Liu, H.: Feature selection for clustering. In: PADKK 2000, pp. 110–121 (2000)

[6] Dhillon, I., Kogan, J., Nicholas, C.: Feature selection and document clustering. In: Survey of Text Mining, pp. 73–100. Springer, Heidelberg (2003)

[7] Gondek, D., Hofmann, T.: Non-redundant data clustering. In: ICDM, pp. 75–82 (2004)

[8] Han, J.: Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco (2005)

[9] Patrikainen, A., Meila, M.: Comparing subspace clusterings. IEEE Transactions on Knowledge and Data Engineering 18(7), 902–916 (2006)

[10] Qi, Z., Davidson, I.: A principled and flexible framework for finding alternative clusterings. In: Elder IV, J.F., Fogelman-Souli, F., Flach, P.A., Zaki, M. (eds.) KDD, pp. 717–726. ACM (2009)

General Spatial Skyline Operator

Qianlu Lin[1], Ying Zhang[1], Wenjie Zhang[1], and Aiping Li[2]

[1] The University of New South Wales Australia
[2] National University of Defense Technology, China
{qlin,yingz,zhangw}@cse.unsw.edu.au, apli1974@gmail.com

Abstract. With the emergence of location-aware mobile device technologies, communication technologies and GPS systems, various location-aware queries have attracted great attentions in the database literature. In many user recommendation systems, the spatial preference query is used to suggest the objects based on their spatial proximity to the facilities. In this paper, we study the problem of *general spatial skyline* which can provide a minimal set of candidates that contain optimal solutions for any monotonic distance based spatial preference query. An efficient algorithm is proposed to significantly reduce the number of non-promising objects in the computation. The paper also covers a comprehensive performance study of the proposed techniques based on both real and synthetic data.

1 Introduction

With the development of mobile device technologies, communication technologies and GPS systems in recent years, there has been an increasing number of location based service systems specialized in providing interesting results through location based queries which retrieve the desirable candidate objects for users based on the spatial proximity of the objects and facilities. For instance, as shown in Figure 1(a), there are a set of apartments, bus stations and supermarkets in the map, and a user wants to rent an apartment which is close to both a bus station and a supermarket. In Figure 1(b), each apartment is mapped to a point in a 2-dimensional space where the distances to the nearest bus station and supermarket are the coordinate values of an apartment. As shown in Figure 1 the apartment a_4 derives its coordinates from the distance to its closest bus station (b_1) and supermarket(s_1). Clearly, the smaller value is preferred. As there is no apartment with both shortest supermarket-distance and bus station-distance in the example, the user needs to make a trade-off. Suppose the user has a preference function against the distances of an apartment regarding its closest bus station and supermarket, the system can return the apartment with best score regarding the preference function. If the preference function is in the form of $f(o) = 4 \times o.d_1 + o.d_2$ where $o.d_1$ and $o.d_2$ represent the distances of o to the closest supermarket and bus station respectively, then a_4 is the best choice. The answer becomes a_3 if we have $f(o) = o.d_1 + 4 \times o.d_2$. This is the distance based spatial preference query[1], and the problem is studied

[1] See Section 2.2 for the formal definition.

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 494–508, 2012.
© Springer-Verlag Berlin Heidelberg 2012

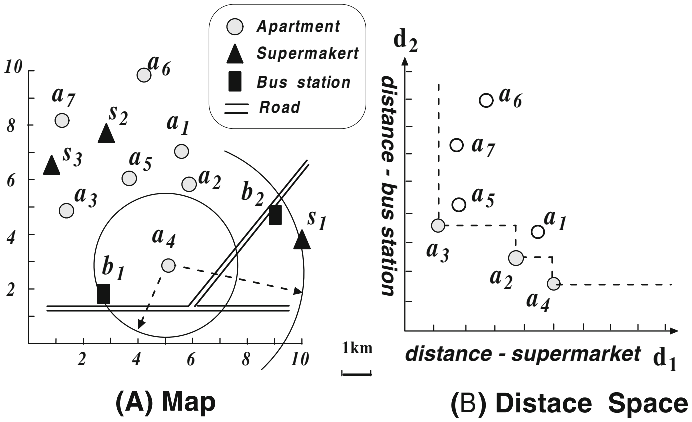


Fig. 1. Motivating Example

in [10,17,12]. However, in many applications users cannot find an appropriate preference function. Therefore, it is desirable to provide a minimal candidate set for users so that they can make personal trade-offs without missing any potential optimal solution.

Motivated by the above example, in this paper we propose the *general spatial skyline* (*GSSKY*) operator. Given a set $\mathcal{O}$ of objects and a set $\mathcal{F}$ of facilities with m types, an object o can be mapped to a point $\tilde{o}$ in m-dimensional space, named distance space, where the coordinate value on i-th dimension is the distance of o to its nearest facility with type i. We say an object o_1 *spatially dominates* another object o_2 if $\tilde{o_1}$ *dominates* $\tilde{o_2}$ in the mapped distance space. Note that the *dominance* relationship in distance space is the same as the traditional skyline problem [1]; that is, we say $\tilde{o_1}$ *dominates* $\tilde{o_2}$ if $\tilde{o_1}$ is not larger than $\tilde{o_2}$ on any dimension $i \in [1, m]$, and $\tilde{o_1}$ is smaller than $\tilde{o_2}$ on at least one dimension. Then the objects which are not *spatially dominated* by any other object are *general spatial skyline* objects. As shown in Section 2.2, the *general spatial skyline* objects can provide a minimal set of candidates that contain optimal solutions for any monotonic distance based spatial preference query. Moreover, we show theoretically and experimentally that the number of *GSSKY* objects is usually much smaller than that of the objects.

Note that although there are some existing works [14,15] which study the problem of *spatial skyline*, they cannot provide a minimal set for the distance based spatial preference queries studied above due to the essential differences between the two problems. Please see Section 6 for detailed discussion.

Challenge. A straightforward solution for the *GSSKY* query is to compute distance values of all objects and then apply the traditional skyline algorithm. This is not efficient because the distance computation (i.e., retrieving the distance of an object to the closest facility regarding a particular type) is expensive and we have to compute the distance values for all objects. In this paper, we propose a novel *GSSKY* computation algorithm which aims to reduce the amount of distance computations by pruning non-promising objects. Our contributions can be summarized as follows.

- The *general spatial skyline* query is formally defined, that provides a minimal set of candidates which contain optimal solutions for any monotonic distance based spatial preference query.
- An efficient algorithm is proposed to compute the *general spatial skyline.*
- Comprehensive experiments demonstrate the efficiency of our techniques.

The remainder of the paper is organized as follows. We formally define the problem and discuss related techniques in Section 2. Section 3 presents the all nearest neighbor based algorithm. Section 4 proposes our efficient *GSSKY* algorithm. Results of a comprehensive performance study are presented in Section 5. Section 6 presents the related work. Finally, Section 7 concludes the paper.

2 Preliminary

In Section 2.1, we formally define the problem of general spatial skyline computation . In Section 2.2, we show that the general spatial skyline can provide a minimal set of candidates that contain optimal solutions for any monotonic spatial preference function. We introduce the incremental nearest neighbor algorithm in Section 2.4. Table 1 below summarizes the mathematical notations frequently used.

Table 1. The summary of notations

Notation	Definition
o ($\mathcal{O}$)	object (a set of objects)
f ($\mathcal{F}$)	facility (a set of facilities)
m	the number of facility types in $\mathcal{F}$
$\mathcal{F}_i$	all facilities in $\mathcal{F}$ with type i
$o.d_i$	the distance between o and its closest facility with type i
$o \lhd \mathcal{F}$	o is *fully hit* by $\mathcal{F}$
r_i	the maximal hit distance seen so far regarding facilities with type i
$o_1 \prec_{\mathcal{F}} o_2$	o_1 *spatially dominates* o_2 regarding $\mathcal{F}$
$GSSKY(\mathcal{O}, \mathcal{F})$	the general spatial skyline of $\mathcal{O}$ regarding the facilities $\mathcal{F}$

2.1 Problem Definition

A point x referred in this paper, by default, is in a d-dimensional numerical space. Let $\delta(x, y)$ denote the Euclidian distance between two points x and y[2]. In the paper, $\mathcal{F}$ represents a set of facilities and $\mathcal{F}_i$ denotes all facilities in $\mathcal{F}$ with type i. A facility f is a point in the space with a particular facility type.

An object o is a point in a d-dimensional numerical space. The distance of o to $\mathcal{F}_i$, denoted by $o.d_i$, is the distance between o and its closest facility with type i, i.e., $o.d_i = \min\ (\ \delta(o, f)$ for any $f \in \mathcal{F}_i)$. As shown in Figure 1, given a

[2] We focus on Euclidian distance in the paper. Nevertheless, our techniques can be easily extended to other L_p norm distances.

set $\mathcal{F}$ of facilities with m categories, an object o can be mapped to a point in m-dimensional space. We define the *spatial dominance* relationship as follows.

Definition 1 (Spatial Dominance). *Given two objects o_1, o_2 and a set $\mathcal{F}$ of facilities, We say object o_1 spatially dominates another object o_2 regarding $\mathcal{F}$, denoted by $o_1 \prec_{\mathcal{F}} o_2$, if and only if $o_1.d_j \leq o_2.d_j$ for* ***any*** *facility type j, and there is a facility type i such that $o_1.d_i < o_2.d_i$.*

Example 1. In Figure 1, we have $a_2 \prec_{\mathcal{F}} a_1$, $a_3 \prec_{\mathcal{F}} a_5$, and $a_4 \not\prec_{\mathcal{F}} a_5$.

Based on the *spatial dominance* relation, we come up with the definition of *general spatial skyline* as follows.

Definition 2 (General Spatial Skyline). *Given a set $\mathcal{O}$ of objects and a set $\mathcal{F}$ of facilities, the general spatial skyline of $\mathcal{O}$ regarding $\mathcal{F}$, denoted by $GSSKY(\mathcal{O},\mathcal{F})$, are objects which are not spatially dominated by any other objects regarding $\mathcal{F}$.*

Example 2. In Figure 1, we have $GSSKY(\mathcal{O},\mathcal{F}) = \{a_2, a_3, a_4\}$.

Problem Statement

In this paper we investigate the problem of efficiently computing general spatial skyline for a set of objects with respective to multiple types of facilities.

2.2 Minimal Set Property

Given a set $\mathcal{O}$ of objects and a set $\mathcal{F}$ of facilities with m types, the score of an object o regarding $\mathcal{F}$, denoted by o_s, is derived based on its closest facilities. Following is a formal definition of the distance based spatial preference query.

$$o_s = \ p(\ o.d_1, \ \ \ldots, o.d_m) \tag{1}$$

Recall that $o.d_i$ denotes the distance between o and its closest facility with type i. For presentation simplicity, we use "spatial preference function" to abbreviate "distance based spatial preference function" in the paper whenever there is no ambiguity. The following theorem indicates that the *GSSKY* provides the minimal set for all increasing spatial preference functions.

Theorem 1. *Let $\mathcal{P}$ denote the family of all increasing spatial preference functions regarding $\mathcal{F}$, for any $p \in \mathcal{P}$ the object with best score is in $GSSKY(\mathcal{O},\mathcal{F})$. For any object o in $GSSKY(o,\mathcal{F})$, there exists a spatial preference function $p \in \mathcal{P}$ such that o has the best score regarding p.*

Proof. For any object $o_2 \notin GSSKY(\mathcal{O},\mathcal{F})$, there is an object o_1 such that $o_1 \in GSSKY(\mathcal{O},\mathcal{F})$ and $o_1 \prec_{\mathcal{F}} o_2$ according to the definition of $GSSKY$. We have $o_1.d_j \leq o_2.d_j$ for any $j \in [1,m]$ and there exists $i \in [1,m]$ such that $o_1.d_i < o_2.d_i$. According to the monotonic property of the functions, we have $p(o_1) < p(o_2)$ for any increasing spatial preference function p. With similar rationale, there is an increasing spatial preference function p for each object $o \in GSSKY(\mathcal{O},\mathcal{F})$ such that $p(o)$ has lowest score among all objects. Therefore, the theorem holds. ■

2.3 Size Estimation

Based on [5], we have the following theorem which estimates the size of *GSSKY* objects using an independence assumption.

Theorem 2. *Suppose the locations of the facilities and the objects are independent to each other, then the expected number of GSSKY object is* $O(\frac{(\ln(n))^{m-1}}{(m-1)!})$ *where* n *is the number of objects in* $\mathcal{O}$.

2.4 Incremental Nearest Neighbor Technique

As our general spatial skyline algorithm proposed in Section 4 is based on the incremental nearest neighbors(INN) computation, we introduce the INN technique [7] in this subsection. Unlike the k nearest neighbor query where k is known beforehand, the INN algorithm will incrementally output the next closest neighbor , i.e., the $(l+1)$-th nearest neighbor where l is the number of neighbors seen so far, on user's demand.

Suppose the objects are organized using an R-tree. A priority queue $\mathcal{Q}$ is used to maintain a set of R-tree entries (intermediate entries and data entries) where the key of an entry is its minimal distance to the query point. The root of the R-tree is pushed into $\mathcal{Q}$ at the beginning of the algorithm. For each incremental nearest neighbor request, the algorithm outputs the data entry in $\mathcal{Q}$ with smallest key value. Note that we say an object is in $\mathcal{Q}$ if its corresponding data entry or any of its ancestor entries is in $\mathcal{Q}$. Specifically, if the entry with smallest key value is a data entry which is associated with an object o, o is output and popped from $\mathcal{Q}$. Otherwise, the intermediate entry (i.e., index or leaf node) is popped and expanded, and all its child entries are pushed into $\mathcal{Q}$. The procedure is repeated until the entry on top of $\mathcal{Q}$ is a data entry. The algorithm can therefore be used to incrementally determine the next nearest neighbor. [7] has shown the efficiency of the INN algorithm theoretically and experimentally.

3 All Neareast Neighbor(ANN) Based GSSKY Algorithm

Since the *GSSKY* problem is exactly the same as the traditional skyline problem if all objects are mapped to the distance space $\mathcal{D}$, a straightforward solution for the *GSSKY* computation is to first compute the distances for all objects regarding $\mathcal{F}$, and then apply the existing skyline algorithm. As the computation of the distance values of the objects regarding facilities with type i can be achieved by all nearest neighbor (ANN) queries against $\mathcal{O}$ and $\mathcal{F}_i$, in this subsection, we apply the state-of-the-art ANN technique [2] to compute the *GSSKY*. Algorithm 1 outlines the ANN based general spatial skyline computation. Note that all existing non-index skyline techniques can be applied in Line 3 once the distance values of all objects are available. As shown in the empirical study, the dominant cost of Algorithm 1 is the distance computation.

Algorithm 1: ***ANN based GSSKY*** $(\mathcal{O}, \mathcal{F})$

```
Input  : $\mathcal{O}$ : the objects,
         $\mathcal{F}$ : the facilities
Output : $\mathcal{S}$ : GSSKY( $\mathcal{O}$, $\mathcal{F}$)
for each facility type $i$ in $[1..m]$ do
    Compute the $o.d_i$ for each object $o \in \mathcal{O}$ by applying ANN [2] algorithm
    against $\mathcal{O}$ and $\mathcal{F}_i$ ;
$\mathcal{S} \leftarrow$ compute skyline on the distances of the objects;
return $\mathcal{S}$
```

4 Efficient *GSSKY* Algorithm

4.1 Motivation

As shown in the empirical study, the dominant cost of the *GSSKY* computation comes from the calculation of the distance values for the objects. Consequently, even if we apply the state-of-the-art technique to compute the distance values for all objects, the ANN based *GSSKY* algorithm is still inefficient in terms of both I/O and CPU costs. Motivated by this, in this section we aim to reduce the number of distance computations during the *GSSKY* query process.

In the paper, we may compute the distance values of the objects in two ways:

Object Oriented Search

For each object o, we compute the $o.d_i$ by applying the nearest neighbor(NN) algorithm [13] where o is the query point. For instance, as shown in Figure 1 $o_4.d_1$ and $o_4.d_2$ can be derived by issuing two NN queries against $\mathcal{F}_1 = \{s_1, s_2, s_3\}$ and $\mathcal{F}_2 = \{b_1, b_2\}$ respectively, where o_4 is the query point. Particularly, the all nearest neighbor (ANN) algorithm [2] can also be considered as an object oriented method in which the object distances are computed in a batch fashion.

The advantage of the object oriented search is that, for a given object or a set of objects, we can directly derive the distances of the objects. However, as there is no a priori knowledge about the distance values of the *unvisited* objects, like Algorithm 1 in Section 3, we have to compute distance values for all objects to ensure the correctness of *GSSKY* computations.

Facility Oriented Search

Instead of computing distance values for each individual object, we can derive them by applying incremental nearest neighbor(INN) algorithm against facilities simultaneously where the query point is a facility. As shown in Figure 2, for each facility $f \in \mathcal{F}$, we maintain a radius f_r and we say an object o has been *hit* by f if $\delta(f, o) \leq f_r$. The distance between o and f is called the hit distance of o regarding f. Similarly, we say an object o is *fully hit* by $\mathcal{F}$, denoted by $o \lhd \mathcal{F}$, if o has been hit by **all types** of facilities; that is, for any $\mathcal{F}_i$, there exists a facility $f \in \mathcal{F}_i$ such that o is *hit* by f. For each facility type i, we maintain a global radius r_i which is the maximal hit distance seen so far regarding facilities with type i.

At each iteration, for each type i we find a facility f in $\mathcal{F}_i$ to invoke a new hit by expanding f_r such that the increment of r_i is minimized. Clearly, the

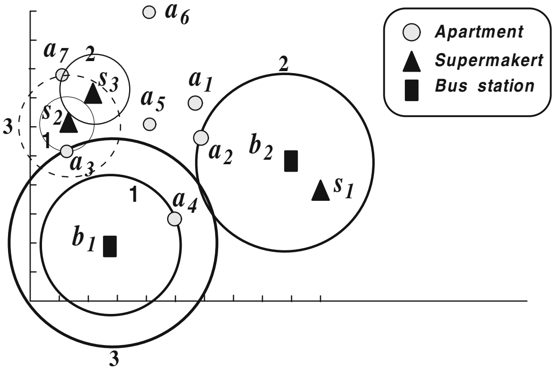


Fig. 2. Running Example

global radius r_i is non-decreasing in the search. Due to the monotonic property of r_i, we can safely set $o.d_i$ to the hit distance when it is *hit* for the first time by a facility with type i. Recall that an object may be *hit* multiple times by the facilities with the same type. Therefore, we say a *hit* is a *redundant hit* if the object has been *hit* by another facility with the same type.

Example 3. Figure 2 illustrates a snapshot of the facility oriented search in which we use a circle to record each hit of the objects. Specifically, circles with thin(bold) line represent the hits from bus stations (supermarkets) and the number of a circle indicates the accessing order. Moreover, the circle with solid (dashed) line represents a *non-redundant hit* (*redundant hit*). In Figure 2, $a_3.d_1$ and $a_4.d_2$ are derived in the first iteration. In the third iteration, the hit of a_7 regarding s_3 is a *redundant hit* because a_7 has been *hit* by s_2 in the second iteration.

Without loss of generality, in the paper we assume the hit distance is distinct for each facility type. Note that the duplication can be easily handled by visiting all objects with the same hit distance. Because of the monotonic property of the hit distance (i.e., r_i), the following lemma is immediate, which enables us to obtain the lower bound of the distance values for the *unvisited* objects.

Lemma 1. *In the facility oriented search, we have $o.d_i > r_i$ if an object o has not been hit by any facility with type i so far.*

Based on Lemma 1, the following theorem implies that we can safely prune some objects from the $GSSKY(\mathcal{O}, \mathcal{F})$ without any distance computation.

Theorem 3. *In the facility oriented search, suppose there exists an object o_1 which has been hit by **all types** of facilities, an object o_2 can be pruned from $GSSKY(\mathcal{O}, \mathcal{F})$ if o_2 has not been hit by **any** facility.*

Proof. We have $o_1.d_i \leq r_i$ for any $i \in [1, m]$ since o_1 has been *hit* by all types of facilities. On the other hand, we have $o_2.d_i > r_i$ for any object o_2 which has not been *hit* by any facility. It is immediate that $o_1 \prec_{\mathcal{F}} o_2$ and hence o_2 can be pruned from $GSSKY(\mathcal{O}, \mathcal{F})$. Therefore, the theorem holds. ■

Example 4. In Figure 2, objects $\{a_1, a_5, a_6\}$ can be pruned from $GSSKY(\mathcal{O}, \mathcal{F})$ without any distance computation because none of them has been *hit* by any facility when a_3 is *fully hit*.

Another advantage of the facility oriented search is that, as shown in [7], the amortized cost for each hit distance computation in INN query is cheaper than that of a NN query because the INN algorithm can share the computation by continuously maintain the priority queue. This implies that if the proportion of the *redundant hits* is not significant, the facility oriented method is more efficient. Intuitively, the proportion of the *redundant hits* will increase with the global radius r_i regarding $\mathcal{F}_i$ because the larger the radius, the higher chance an object is *hit* by multiple facilities in $\mathcal{F}_i$. Another disadvantage of the the facility oriented search is that we need to maintain a priority queue for each facility and it is not space efficient when the number of facilities is very large.

Motivated by the advantages and disadvantages of the object oriented search and the facility oriented search, we propose an efficient *GSSKY* algorithm which combine both methods in an effective way. The algorithm consists of three phases. In the first phase, we apply the facility oriented search to compute object distances and prune objects (i.e., remove non-skyline objects) based on lemma 1 and Theorem 3. This is feasible because the number of facilities is usually much smaller than that of objects in real applications. When we find that the computation of facility oriented search becomes less efficient due to the large amount of *redundant hits*, the algorithm goes to phase two, in which we compute the distances of the remaining objects based on the object oriented search (i.e., NN query). Finally, in phase three we apply the existing skyline algorithm to finalize the *GSSKY* computation.

4.2 Algorithm

In the paper, we assume a set $\mathcal{O}$ of objects are organized using R-Tree, denoted by $R_{\mathcal{O}}$, and all facilities with type i are also organized using R-Tree $R_{\mathcal{F}_i}$. The Algorithm 2 illustrates the details of the efficient *GSSKY* algorithm.

In Line 2-9, we apply the facility oriented search to compute the distances of the objects until there exists an object which has been *fully hit*. Particularly, a *local priority queue* is employed for each facility f for INN query, i.e., retrieve the next closest neighbor of f. For each facility type $i \in [1, m]$, we use a *global priority queue* $\mathcal{Q}_i$ to maintain the current closest neighbors (i.e., objects) of the facilities in $\mathcal{F}_i$. The elements *global priority queue* $\mathcal{Q}_i$ are prioritized using distances. In Line 4, the object o on the top of $\mathcal{Q}_i$ is popped and a INN query is issued by its associated facility to retrieve the next closest neighbors o_2. Then o_2 is pushed into $\mathcal{Q}_i$. Line 6 sets $o.d_i$ to the current hit distance (recorded by r_i) if it is a *non-redundant hit*. When the loop is terminated (Line 9), o is a *GSSKY* object and kept in $\mathcal{S}$, and objects which have been *hit* at least once are kept in the candidate set $\mathcal{C}$. According to Theorem 3, all remaining objects can be pruned. For I/O efficiency, we keep the page ids of the nodes (i.e., intermediate entries) of the $R_{\mathcal{O}}$ visited so far. In the following facility oriented search (Line 10- 21), we do not access a node of $R_{\mathcal{O}}$ if its page id is not recorded

Algorithm 2: Efficient *GSSKY* Algorithm ($\mathcal{O}$, $\mathcal{F}$)

```
   Input   : $\mathcal{O}$ : the objects,
             $\mathcal{F}$ : the facilities with $m$ types
   Output  : $\mathcal{S}$ : GSSKY( $\mathcal{O}$, $\mathcal{F}$)
 1 $\mathcal{S} := \emptyset$; $\mathcal{C} := \emptyset$; $r_i := 0$ for each $\mathcal{F}_i$;
 2 while true do
 3     for each facility type $i$ in $[1..m]$ do
 4         $o \leftarrow$ next object in facility oriented search regarding $\mathcal{F}_i$;
 5         if the hit of $o$ is a non-redundant hit then
 6             $o_{d_i} := r_i$; $\mathcal{C} := \mathcal{C} \cup o$;
 7         if $o$ is fully hit by $\mathcal{F}$ then
 8             $\mathcal{S} := o$; $\mathcal{C} := \mathcal{C} - o$;
 9             Terminate the while loop;
10 while true do
11     for each facility type $i$ in $[1..m]$ do
12         $o \leftarrow$ next object in facility oriented search regarding $\mathcal{F}_i$;
13         if $o$ is a candidate object and the hit of $o$ is a non-redundant hit then
14             $o_{d_i} := r_i$;
15             if SkylineTest($\mathcal{S}$, $o$) then
16                 if $o$ is fully hit by $\mathcal{F}$ then
17                     $\mathcal{C} := \mathcal{C} - o$; $\mathcal{S} := \mathcal{S} + o$;
18             else
19                 $\mathcal{C} := \mathcal{C} - o$;
20         if $\#$redundant hit is larger than $\#$non-redundant hit then
21             Terminate the while loop;
22 for each object $o \in \mathcal{C}$ do
23     calculate $o_{d_i}$ by NN query if $o$ has not been hit regarding $\mathcal{F}_i$ ;
24 for each object $o \in \mathcal{C}$ accessed in non-decreasing order based on $\sum_{i=1}^{m} o_{d_i}$ do
25     if SkylineTest($\mathcal{S}$, $o$) then
26         $\mathcal{S} := \mathcal{S} \cup o$;
27 return $\mathcal{S}$
```

(i.e., all of its descendant data entries correspond to the pruned objects) and hence the I/O cost can be saved.

In Line 10-21, we continue the facility oriented search and try to identify the *GSSKY* objects and prune the non-promising ones. Particularly, if the object o output in Line 12 is a candidate object (i.e., $o \in \mathcal{C}$) and the hit is a *non-redundant hit*, Line 15 checks if there exits an object $s \in \mathcal{S}$ (i.e., *GSSKY* objects seen so far) such that $s \prec_{\mathcal{F}} o$. Note that if o has not been *hit* by any facility with type i, o_{d_i} is temporarily set to r_i in the test. We say an object o passes the skyline test (**SkylineTest**) if it is not *spatially dominated* by any object in $\mathcal{S}$. In the case o passes the test (Line 4.2-17), it is a *GSSKY* object **if** o has been *fully hit*. Otherwise, we cannot claim that o is a *GSSKY* object at this moment because the lower bound of the distance is employed in the skyline test. Line 19 eliminates the object from $\mathcal{C}$ if o fails the test. As discussed in Section 4.1, the facility oriented seasrch should be stopped when r_i becomes large due to the increased probability of *redundant hit*. However, it is impossible to find the optimal stop time without knowing the exact distributions of the following *redundant hits* and *non-redundant hits* are unknown. In the paper, we employ a simple but effective criteria. The number of *non-redundant hits* and *redundant hits* are counted, and Line 21 terminates the facility oriented search if there are more *redundant hits*.

Line 22-23 calculate the missing distances for the objects in the candidate set, where the object oriented search is employed. Recall that the missing of o_{d_i} value implies that the object o has not been *hit* by any facility with type i so far. The remaining part of the algorithm is similar to the SFS Algorithm [3]. Particularly, we sort all the candidate objects based on the sum of their distance values (i.e., $\sum_{i=1}^{m} o_{d_i}$) in *non-decreasing* order (Line 22), and Line 25 checks if an object is *spatially dominated* by the *GSSKY* objects ($\mathcal{S}$) seen so far. The objects passed the test are *GSSKY* objects (Line 26).

Correctness. For the correctness of the Algorithm 2, we need the following properties: (i) any object pruned at Line 15 and Line 25 are not *GSSKY* object, (ii) all objects *unvisited* in Algorithm 2 are not *GSSKY* objects, and (iii) the object in $\mathcal{S}$ cannot be dominated by any other object in $\mathcal{O}$. Below is a formal proof.

Proof. The correctness of the property (i) is immediate based on the definition of *GSSKY* if o has been *fully hit* at Line 15 and Line 25. If o_{d_i} is replaced by r_i at Line 15 (i.e., o has not been *hit* by any facility with type i) and o is dominated by an object $s \in \mathcal{S}$, we can claim that s *spatially dominates* o regarding $\mathcal{F}$ due to the monotonic property of r_i (Lemma 1). The correctness of property (ii) is immediate based on Theorem 3.

We prove the correctness of property (iii) by the contradiction. Suppose the object s is in $\mathcal{S}$ but s is *spatially dominated* by another object o. We can assume o is a *GSSKY* object because of the *dominance transitivity* property of *spatial dominance*, i.e., $o_1 \prec_{\mathcal{F}} o_2$ and $o_2 \prec_{\mathcal{F}} o_3$ implies $o_1 \prec_{\mathcal{F}} o_3$. If s is put in $\mathcal{S}$ at Line 4.2 or Line 17, o should be included in $\mathcal{S}$ before s. This is because $o \prec_{\mathcal{F}} s$ implies s is *fully hit* after o due to the monotonic property in the facility oriented search. This contradicts the proposition that s is the first object being *fully hit* (). Also, s should also fail in the test in and s will not be added into $\mathcal{S}$. We can come up with similar contradiction if s is put in $\mathcal{S}$ at Line 26 because we access objects based the sum of their distance values. ■

Performance Analysis. Upon each hit in Algorithm 2 (Line 2-21), an INN query is issued to retrieve the next closest neighbor of a facility $f \in \mathcal{F}_i$ where $i \in [1, m]$ and the global priority queue $\mathcal{Q}_i$ is updated. The cost is $C_{inn} + O(\log(n_f))$ where C_{inn} and n_f denote the average cost of a INN query and the average number of facilities for each type respectively. If it is a *non-redundant hit*, the skyline test is invoked which costs $O(|\mathcal{S}|)$ in the worst case where $|\mathcal{S}|$ is the size of $\mathcal{S}$. In Line 22-23, the cost is $O((m-1) \times |\mathcal{C}| \times C_{nn})$ in the worst case where C_{nn} is the average cost for NN query and $|\mathcal{C}|$ denotes the candidate set size. Recall that a candidate object will be *hit* at least once. The sorting cost in Line 24 is $O(|\mathcal{C}| \times \log(|\mathcal{C}|))$ and the cost of skyline computation in Line 24-26 is $|\mathcal{S}|^2$ in the worse case. In summary, let n_r and n_s denote the number of *redundant hits* and *non-redundant hits*, the time complexity of Algorithm 2 is $O((n_r+n_s)\times(C_{inn}+\log(n_f)) + n_s\times|\mathcal{S}| + (m-1)\times|\mathcal{C}|\times C_{nn} + |\mathcal{C}|\times\log(|\mathcal{C}|)+|\mathcal{S}|^2)$. Note that, in practice $|\mathcal{S}|$ and $|\mathcal{C}|$ are much smaller than the total number of objects, and hence the algorithm is quite efficient.

Following theorem estimate the number of objects accessed, i.e., objects have been *hit* at least once, in Algorithm 2 based on the uniform and independence assumption.

Theorem 4. *Suppose that (i) objects and facilities are uniformly distributed in the space* $[0,1]^2$*, (ii) there are* n_f *facilities for each type, and (iii) the locations are independence regarding different types of facilities. The expected number of objects accessed in Algorithm 2 is* $n(1-(1-\pi\bar{X}^2)^m)$ *where* n *is the number of objects. Particularly, we have* $\bar{X}$ *equals* $\int_{r=0}^{c}(1-F(r))'r\ d(r)$ *where* $c = \frac{1}{\sqrt{2}n_f}$*, and* $F(r) = (1-(n_f\pi r^2)^m)^n$*.*

Proof. Due to the space limitation, we give a brief proof. According to the uniform assumption and each type has the same number of facilities, we can assume $r_i = r_j$ in each iteration where $1 \leq i, j \leq m$. Therefore, we use r to denote the r_i for any $i \in [1, m]$. The probability that none of the objects is fully *hit* for given r, denoted by $F(r)$, is $(1-(n_f\pi r^2)^m)^n$ due to the uniform and independence assumption. Let X denote the distance r when the first object is fully *hit*, then its expected value $\bar{X}$ equals $\int_{r=0}^{c}(1-F(r))' \times r\ d(r)$ where $c = \frac{1}{\sqrt{2}n_f}$. Consequently, the expected number of objects accessed is $n(1-(1-\pi\bar{X}^2)^m)$. ■

5 Performance Evaluation

In this section, We present the results of a comprehensive performance study to evaluate the efficiency and scalability of the proposed techniques in the paper. The following algorithms were selected for evaluation.

***ANN*.** The all nearest neighbor based technique presented in Section 3. The SFS algorithm [3] is used in Algorithm 1 for skyline computation.

***GSSKY*.** The efficient *GSSKY* algorithm proposed in Section 4.

Both algorithms in this paper are implemented in standard C++ with STL library support and compiled with GNU GCC. Experiments are run on a PC with Intel Xeon 2.40GHz dual CPU and 4G memory running Debian Linux. The disk page size is fixed to 4096 bytes.

Real Datasets. Two real spatial datasets, *CA* and *US*, are employed in the experiment[3]. *CA* consists of 104,217 locations of 44 different categories (e.g., church, lake and school). Each category corresponds to a facility type. The objects in *CA* are constructed as follows. We first normalize the space to $[0,1]^2$ and

[3] *CA* is available from http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm. *US* is available from http://www.geonames.usgs.gov/

then for each facility we randomly create 5 objects within distance 0.005. Consequently the number of objects in *CA* dataset is $521,085$. Similarly, *US* dataset is obtained from the U.S. Geological Survey (USGS) and consists of $406,709$ locations with 40 types. The number of objects in *US* is $2,033,545$.

Synthetic Datasets. To study the scalability of the algorithms, we also create synthetic dataset, denoted by SYN, in the experiment. The objects and facilities are randomly generated in 2-dimensional space $[0,1]^2$. Specifically, the number of objects varies from $500K$ to $5M$ with default value $1M$. There are 40 types of facilities and the number of facilities for each type varies from 500 to $10,000$ with default value $2,000$. SYN is the default dataset in the experiment.

Work Load. The work load of each experiment consists of 200 $GSSKY$ queries and m types are randomly chosen in each query where m varies from 2 to 5 with default value 3. In the paper, the average processing time, which includes the CPU time and I/O latency, is used to measure the efficiency of the algorithms. We also record the average $GSSKY$ size and the average number of nodes loaded.

Table 2 lists parameters which may have an impact on our performance study. In our experiments, all parameters use default values unless otherwise specified.

Table 2. System Parameters

Notation	**Definition (Default Values)**
m	the number of facility types (3)
n	the number of objects ($1M$)
n_f	the number of facilities for each type (2000)

5.1 *GSSKY* SIZE

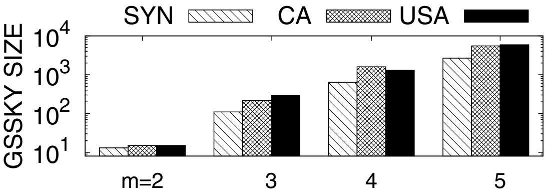


Fig. 3. Diff. datasets and m

In this subsection, we investigate the size of the $GSSKY$. Figure 3 illustrates the $GSSKY$ size on SYN, *CA* and *US* datasets where m varies from 2 to 5. For given m, the size difference of the three datasets are not significant. As expected, the number of $GSSKY$ objects increases quickly towards the number of types (m). Particularly, for $m = 2$ the $GSSKY$ size is 13, 15 and 15 for SYN, *CA* and *US* respectively. When m goes to 5, it becomes $2,666$, $5,508$ and $5,911$ respectively.

Figure 4 and Figure 5 investigate the impact of the object size (n) and facility size (n_f) respectively. Since locations of the facilities with different types are independent, Theorem 2 can be applied to estimate the $GSSKY$ size , and its accuracy is verified in both Figures. Moreover, the $GSSKY$ size increases slowly with the number of objects and is independent to the number of facilities.

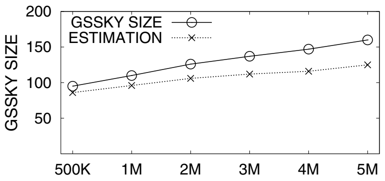


Fig. 4. Diff. n

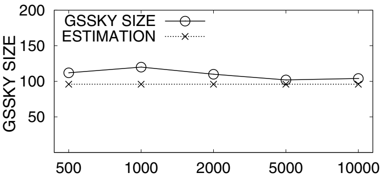
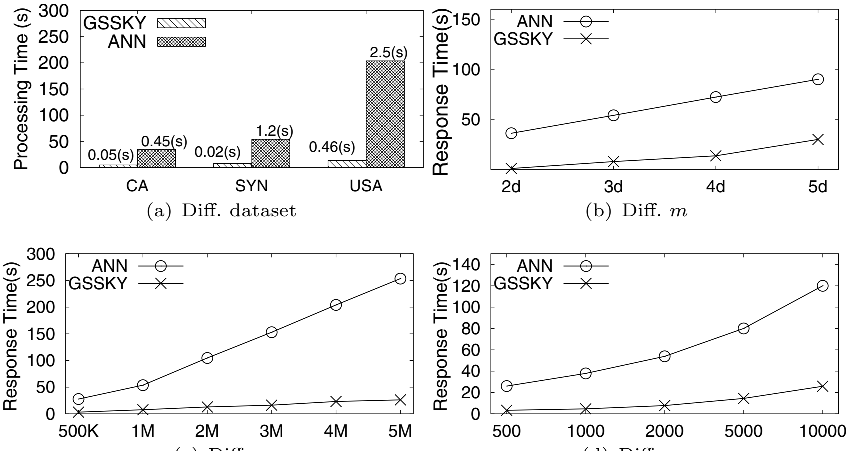


Fig. 5. Diff. n_f

5.2 Efficiency

We first evaluate the efficiency of the algorithms on SYN, CA and US datasets. Figure 6(a) shows that $GSSKY$ Algorithm significantly outperforms the ANN Algorithm by at least one order of magnitude. The number on each bar records the cost for the skyline test, which shows the dominant cost in two algorithms is the distance computation. We also study the impact of the parameters which may potentially affect the performance of the algorithms. Specifically, Figure 6(b), Figure 6(c) and Figure 6(d) investigate the scalability of the algorithms against the m (#types), n (#objects) and n_f (#facilities each type) respectively. As expected, the performance of the algorithms degrades against the growth of these parameters. Nevertheless, $GSSKY$ Algorithm is more scalable than ANN Algorithm against n and n_f. As expected, both algorithms are sensitive to m as the $GSSKY$ size increases significantly against m.



(a) Diff. dataset

(b) Diff. m

(c) Diff. n

(d) Diff. n_f

Fig. 6. Time Efficiency Evaluation

In Figure 7, we evaluate the number of R-tree nodes loaded in the main memory on SYN,CA and US datasets. It is shown that $GSSKY$ algorithm significantly reduces I/O because many objects are pruned. Figure 8 shows the proportion of the objects involved in distance computation. Clearly, all objects contribute to the distance calculation in ANN Algorithm. While a significant number of objects are pruned in $GSSKY$ Algorithm. It also demonstrates the accuracy of the Theorem 4, where EST represents the estimation of the theorem.

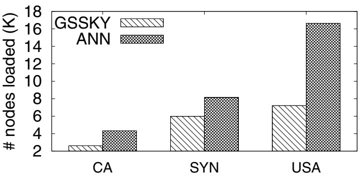


Fig. 7. I/O evaluation

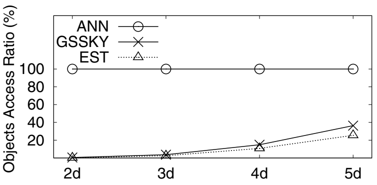


Fig. 8. Object accessed vs m

6 Related Work

Studies on skyline computation have a long history. Börzsönyi *et al.* [1] first investigate the skyline computation problem in the context of databases and propose an SQL syntax for the skyline query. They also develop skyline computation techniques based on *block-nested-loop* and *divide-conquer* paradigms, respectively. Chomicki *et al.* [3] propose another block-nested-loop based computation technique, SFS (*sort-filter-skyline*), to take advantages of a pre-sorting. Papadias *et al.* [11] propose a branch and bound search technique (BBS) to progressively output skyline points on dataset indexed by R-tree.

The problem of spatial skyline is first proposed in [14]. Given a set $\mathcal{O}$ of objects and a set $\mathcal{Q}$ of query points, each object has $|\mathcal{Q}|$ derived spatial attributes each of which is the distance of the object to a query point in $\mathcal{Q}$, and hence can be mapped to a point in $|\mathcal{Q}|$-dimensional space where $|\mathcal{Q}|$ is the number of query points in $\mathcal{Q}$. Then the spatial skyline regarding $\mathcal{O}$ and $\mathcal{Q}$ is the traditional skyline on $|\mathcal{Q}|$-dimensional space. Efficient algorithms are developed in [14] to compute spatial skylines by utilizing the R-tree, convex hull, and voronoi diagram techniques. Son *et al.* [15] further improve the spatial skyline computation techniques. Recently, in [16] they investigate the problem based on the manhattan distance. In [4], Ke *et al.* investigate the problem in the road network. Besides the spatial skyline, there are also some related works in which the skyline is computed based on the derived spatial attributes. In [8], Huang *et al.* studies the problem of in-route skyline to find locations which are not dominated by other candidate locations regarding the network distance to a query location q and the corresponding detour distance. In [9,6] spatial distance regarding a query point q is considered during the skyline computation, in which other dimensions of an objects are non-spatial attributes.

In many applications, the query points may come from the same category (e.g., bus stations, supermarkets). For an object o and a particular category (i.e., facility type like bus station), users are only interested in the distance between o and its closest query point (i.e., facility) in that category. Consequently, the spatial skyline does not make sense in these applications because it considers the distances of o regarding **all** facilities in the same category, and hence cannot provide a minimal set of candidates for the distance based spatial preference queries [10]. Moreover, the techniques in [14,16] cannot be applied to the *GSSKY* computation because the *spatial skyline* is a special case of the *general spatial skyline* in which there is only one facility for each facility type.

7 Conclusion and Future Work

In this paper, we introduce the *general spatial skyline* which can provide a minimal set of candidates that contain optimal solutions of any monotonic distance based spatial preference query. Efficient algorithm is proposed in the paper and comprehensive experiments are conducted to demonstrate the effectiveness and efficiency of the algorithms. As a possible future work, we will investigate the problem on the road network in which the network distance is considered.

Acknowledgement. The work was supported by ARC grants(DP110104880, DP120104168, DE120102144) and ECR grant(PSE1799).

References

1. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE (2001)
2. Chen, Y., Patel, J.M.: Efficient evaluation of all-nearest-neighbor queries. In: ICDE (2007)
3. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: ICDE (2003)
4. Deng, K., Zhou, X., Shen, H.T.: Multi-source skyline query processing in road networks. In: ICDE (2007)
5. Godfrey, P.: Skyline Cardinality for Relational Processing. In: Seipel, D., Turull-Torres, J.M.a. (eds.) FoIKS 2004. LNCS, vol. 2942, pp. 78–97. Springer, Heidelberg (2004)
6. Guo, X., Ishikawa, Y., Gao, Y.: Direction-based spatial skylines. In: MobiDE (2010)
7. Hjaltason, G.R., Samet, H.: Distance browsing in spatial databases. TODS 24(2), 265–318 (1999)
8. Huang, X., Jensen, C.S.: In-Route Skyline Querying for Location-Based Services. In: Kwon, Y.-J., Bouju, A., Claramunt, C. (eds.) W2GIS 2004. LNCS, vol. 3428, pp. 120–135. Springer, Heidelberg (2005)
9. Kodama, K., Iijima, Y., Guo, X., Ishikawa, Y.: Skyline queries based on user locations and preferences for making location-based recommendations. In: GIS-LBSN (2009)
10. Lin, Q., Xiao, C., Cheema, M.A., Wang, W.: Finding the Sites with Best Accessibilities to Amenities. In: Yu, J.X., Kim, M.H., Unland, R. (eds.) DASFAA 2011, Part II. LNCS, vol. 6588, pp. 58–72. Springer, Heidelberg (2011)
11. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: SIGMOD 2003 (2003)
12. Rocha-Junior, J.B., Vlachou, A., Doulkeridis, C., Nørvåg, K.: Efficient processing of top-k spatial preference queries. PVLDB 4(2) (2010)
13. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: SIGMOD Conference (1995)
14. Sharifzadeh, M., Shahabi, C.: The spatial skyline queries. In: VLDB (2006)
15. Son, W., Lee, M.-W., Ahn, H.-K., Hwang, S.-w.: Spatial Skyline Queries: An Efficient Geometric Algorithm. In: Mamoulis, N., Seidl, T., Pedersen, T.B., Torp, K., Assent, I. (eds.) SSTD 2009. LNCS, vol. 5644, pp. 247–264. Springer, Heidelberg (2009)
16. Son, W., Hwang, S.-w., Ahn, H.-K.: MSSQ: Manhattan Spatial Skyline Queries. In: Pfoser, D., Tao, Y., Mouratidis, K., Nascimento, M.A., Mokbel, M., Shekhar, S., Huang, Y. (eds.) SSTD 2011. LNCS, vol. 6849, pp. 313–329. Springer, Heidelberg (2011)
17. Yiu, M.L., Dai, X., Mamoulis, N., Vaitis, M.: Top-k spatial preference queries. In: ICDE (2007)

Top-k Similarity Join over Multi-valued Objects

Wenjie Zhang, Jing Xu, Xin Liang, Ying Zhang, and Xuemin Lin

University of New South Wales, Sydney, NSW, Australia
{zhangw,xjing,xinliang,yingz,lxue}@cse.unsw.edu.au

Abstract. The top-k similarity joins have been extensively studied and used in a wide spectrum of applications such as information retrieval, decision making, spatial data analysis and data mining. Given two sets of objects $\mathcal{U}$ and $\mathcal{V}$, a top-k similarity join returns k pairs of most *similar* objects from $\mathcal{U}\times\mathcal{V}$. In the conventional model of top-k similarity join processing, an object is usually regarded as a point in a multi-dimensional space and the similarity between two objects is usually measured by distance metrics such as Euclidean distance. However, in many applications an object may be described by multiple values (instances) and the conventional model is not applicable since it does not address the distributions of object instances. In this paper, we study top-k similarity join queries over multi-valued objects. We apply quantile based distance to explore the relative instance distribution among the multiple instances of objects. Efficient and effective techniques to process top-k similarity joins over multi-valued objects are developed following a filtering-refinement framework. Novel distance, statistic and weight based pruning techniques are proposed. Comprehensive experiments on both real and synthetic datasets demonstrate the efficiency and effectiveness of our techniques.

1 Introduction

Given two sets of objects (points) $\mathcal{U}$ and $\mathcal{V}$ in a d-dimensional metric space, the top-k similarity join query retrieves k pairs of objects $\mathcal{P}$ from $\mathcal{U} \times \mathcal{V}$ such that the distance between any pair of objects in $\mathcal{P}$ is not greater than the distance of any object pairs in $\mathcal{U} \times \mathcal{V} - \mathcal{P}$. Conventional similarity join query has been extensively studied in various applications including data mining, information retrieval, and location based services [2], [9], [10]. Top-k similarity join, also called *closest pair queries*, has also attracted much research attention [6]. In many applications such as decision making and e-business, an object may be represented by multiple points (instances) in the d-dimensional space, namely *multi-valued* objects [7]. In this paper, we study the problem of top-k similarity joins on *multi-valued* objects.

The needs of similarity join over multi-valued objects stem from many important applications. In geographic information system (GIS), a group of simple spatial objects may be evaluated as a whole [17]. For instance, to evaluate a community, a real estate development company may model it as a multi-valued object and each instance corresponds to a property with some feature values such as property price, household income, distance to beach, distances to living

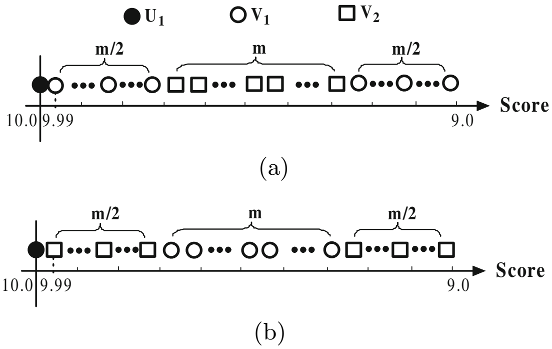
S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 509–525, 2012.
© Springer-Verlag Berlin Heidelberg 2012

facilities, etc. A top-k similarity join may be issued to identify the most similar communities from two large cities or from two countries, such that the price fluctuation of one community could be used as a mirror to the management of another one. Similarly, in sports, the performance of a player may be described by her game-to-game statistics in various games. So each player could be represented by a multi-valued object where each instance corresponds to her statistics, such as heights and number of trials in high-jump, in a particular game she attended. A similarity join over two sets of players may help to retrieve players with similar performances. Hence, the successful career path of one player give a prediction of the success of her counterpart in coming competitions.

While the similarity between two conventional d-dimensional objects only involves two single points, identifying the most similar object pairs among multi-valued object sets involves multiple instances per object. Therefore, it is highly desirable to consider the relative instance distributions among multi-valued objects so that the similar pairs can be effectively retrieved. In this paper, we investigate the problem of similarity join over multi-valued objects in a top-k fashion. That is, we aim to retrieve k pairs of multi-valued objects with the highest level of similarity.

The existing model for handling similarity joins over objects with multiple instances follows the probabilistic semantics on uncertain objects [4], [11], [14] and aims to capture relative instance distribution among objects with multiple instances. Nevertheless, uncertain objects are inherently different than multi-valued objects. Instances of an uncertain object are mutually exclusive which means at most one instance can appear at a particular time, while all the values/instances of a multi-valued object must occur simultaneously at any time. Moreover, as shown in [21], models based on uncertain semantics cannot always capture the relative distributions of multi-valued objects. Take the example in Figure 1. For simplicity we assume multi-valued object U_1 has only one instance with the value (score) of 10, while multi-valued objects V_1 and V_2 both have m instances spread between 9.0 to 9.99 as depicted in Figure 1(a). Each instance from the same object takes the same weight. Suppose we want to retrieve the top-1 similarity join result from $\{U_1\}$ and $\{V_1, V_2\}$, namely, retrieve the more similar one from V_1 and V_2 to U_1. Following the possible world semantics, it is easy to verify that both V_1 and V_2 have the same probability, $\frac{1}{2}$, to be the most similar one to U_1 if Euclidean distance is used as the similarity metrics. We permute the distribution in Figure 1(a) to the distribution in Figure 1(b), V_1 and V_2 still have the same probability. This example demonstrates that the probabilistic approaches following the possible world semantics are not able to capture the relative distributions of instances. Another simple solution is to utilize simple aggregates such as average. Nevertheless, such a simple aggregate will have the same problem as pointed above regarding Figure 1.

The example in Figure 1 demonstrates that the existing probabilistic model and simple aggregates may be insensitive to relative distributions of object instances. Quantiles [19] provide a succinct summary of data distributions. In this paper, we investigate the top-k similarity join problem over multi-valued objects based on a



(a)

(b)

Fig. 1. Motivating Example

ϕ-quantile distance ($\phi \in (0, 1]$); for example, median is the 0.5-quantile, maximum is the 1-quantile, minimum is the smallest quantile (note a quantile ϕ is in $(0, 1]$ and cannot be 0). Regarding the above example, 0.5-quantile is based on players' median performance; 1-quantile is to retrieve the top-k similar pairs based on players' worst performance. In this paper, we study the problem of top-k similarity joins over multi-valued objects where the input are two sets of multi-valued objects.

Challenges and Contributions. To the best of our knowledge, this is the first paper to study top-k similarity joins over multi-valued objects. ϕ-quantile distance is first used for capturing instance distributions of multi-valued objects in [21]. [21] studies top-k nearest neighbor (KNN) queries over multi-valued objects. Given a multi-valued query object Q and a set of multi-valued objects $\mathcal{U}$, a KNN query retrieve k objects from $\mathcal{U}$ with smallest quantile-based distance to Q. An immediate way to solve our problem can be conducted as follows. For each object $U \in \mathcal{U}$ (or $V \in \mathcal{V}$), we compute its KNN in $\mathcal{V}$ (or $\mathcal{U}$) using the techniques in [21], and then select k most similar pairs based on the union of KNN results. Nevertheless, this involves the computation of KNN for each object in $\mathcal{U}$ (or each object in $\mathcal{V}$). Clearly, not every object in $\mathcal{U}$ (or $\mathcal{V}$) will be involved in the top-k pairs since k is usually much smaller than $min\{|\mathcal{U}|, |\mathcal{V}|\}$. Motivated by this, in this paper, we present a set of novel, efficient, effective pruning techniques to prevent such redundant computation. Our main contributions of the paper can be summarized as follows.

- We formalize the problem of top-k similarity join over multi-valued objects, regarding quantile-based distance metrics.
- Efficient and effective algorithms are developed to compute the top-k similarity join results over two sets of multi-valued objects based on ϕ-quantile-based distances. Particularly, we propose novel and efficient distance, statistic and weight based pruning techniques to significantly speed up the computation.
- Comprehensive experiments are conducted on both real and synthetic data to demonstrate the efficiency and effectiveness of our techniques. It also

demonstrates that the techniques developed in this paper are up to 2 orders of magnitude more efficient than naively applying KNN techniques in [21].

Organization of the Paper. The rest of the paper is organized as follows. Section 2 formally defines the problem of top-k similarity join over multi-valued objects regarding quantile-based distance and provide some necessary background information. In Section 3, we introduce the filtering-refinement framework, as well as the data structures utilized in the paper. Section 4 presents query processing techniques for top-k similarity joins. In Section 5, we report our experiment results. Related work is summarized in Section 6. This is followed by conclusions in Section 7.

2 Background

We present problem definition and necessary preliminaries in this section. For references, notations frequently used in the paper are summarized in Table 1.

Table 1. The summary of Notations

Notation	**Definition**
$\mathcal{U}$, $\mathcal{V}$	two sets of of objects in the join query
U (V)	multi-valued object
E	entry of R-tree
u (v)	instance of U (V) - a point in d-dimensional space
$w(u)$ $(w(S))$	(total) weight of u (the set S)
$d(u, v)$	Euclidean distance between u and v
$d^{lo}(E, E')$	distance lower-bound between E and E'
$d_\phi(U, V)$	ϕ-quantile distance of U and V
$U \times V$	Cartesian product of instances from U to V

2.1 Problem Definition

Multi-valued Object. In our problem definition, an instance of an object U is weighted - weight gives the *representativeness* of an instance in U. For instance, in the examples in Section 1, a game statistic of a player may appear multiple times; consequently a normalized weight (the occurrence of an instance over the total occurrences of all instances) may be used to indicate the representativeness of an instance. Note that the total of such weights in U equals to 1.

A multi-valued object U is represented as $\{(u_i, w(u_i)) | 1 \leq i \leq m\}$ where u_i is a point in a d-dimensional space, $0 < w(u_i) \leq 1$ $(1 \leq i \leq m)$, and $\sum_{i=1}^{m} w(u_i) = 1$. We use $\mathcal{U}$ and $\mathcal{V}$ to denote two sets of multi-valued objects involved in the join query.

Quantile. Given a collection S of m elements, each element s_i has a weight $w(s_i)$ where $0 < w(s_i) \leq 1$ and $\sum_{i=1}^{m} w(s_i) = 1$. Let S be sorted increasingly on a search key f - a function; that is, $f(s_i) \leq f(s_j)$ if $i < j$.

Definition 1 (ϕ-quantile of S). *Given a ϕ ($0 < \phi \leq 1$), the ϕ-quantile S_ϕ of S is the* **first** *element s_i in the* **sorted** *S on the search key such that $\sum_{j=1}^{i} w(s_j) \geq \phi$.*

ϕ-quantile Distance. For two given objects U and V, there are totally ($|U| \times |V|$) pairs of instances in $U \times V$ where each pair (u_i, v_j) ($u_i \in U$ and $v_j \in V$) has the weight $w(u_i) \times w(v_j)$, namely $w(u_i, v_j)$. Clearly, $\sum_{u_i \in U, v_j \in V} w(u_i) \times w(v_j) = 1$. The *Euclidean distance* $d(u_i, v_j)$[1] between u_i and v_j is called the distance of (u_i, v_j). Let $U \times V = \{((u_i, v_j), w(u_i, v_j)) \mid u_i \in U \;\&\; v_j \in V\}$.

Definition 2 (ϕ-quantile distance of U and V). *Given a $\phi \in (0, 1]$, let $U \times V$ be sorted increasingly on the search key - the distance $d(u_i, v_j)$ of each element (u_i, v_j). Then, the distance of the ϕ-quantile of $U \times V$ is called the ϕ-*quantile distance *of $U \times V$, denoted by $d_\phi(U, V)$.*

Definition 2 states that if (u, v) is the ϕ-quantile of $U \times V$ (i.e., $(U \times V)_\phi = (u, v)$) then $d(u, v)$ is $d_\phi(U, V)$.

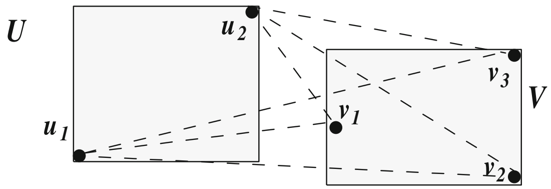


Fig. 2. Distances between 2 Multi-Valued Objects

Example 1. Regarding the example in Figure 2, $|U| = 2$ and $|V| = 3$. Assume that $w(u_1) = w(u_2) = \frac{1}{2}$; $w(v_1) = w(v_2) = \frac{1}{4}$, $w(v_3) = \frac{1}{2}$. Consequently, $U \times V$ consists of the following six instance pairs sorted on their distances increasingly: $U \times V = \{((u_2, v_1), \frac{1}{8}), ((u_2, v_3), \frac{1}{4}), ((u_1, v_1), \frac{1}{8}), ((u_2, v_2), \frac{1}{8}), ((u_1, v_2), \frac{1}{8}), ((u_1, v_3), \frac{1}{4})\}$. The 0.2-quantile distance $d_{0.2}(U, V)$ of U and V is $d(u_2, v_3)$, $d_{0.5}(U, V)$ is $d(u_1, v_1)$, $d_{0.6}(U, V)$ is $d(u_2, v_2)$. □

Problem Statement. Given a $\phi \in (0, 1]$, two sets of multi-valued objects $\mathcal{U}$ and $\mathcal{V}$ in the d-dimensional space, a top-k similarity join retrieves k pairs of objects $\mathcal{P}$ from $\mathcal{U} \times \mathcal{V}$ such that for each object pair (U, V) from $\mathcal{P}$, its ϕ-quantile distance $d_\phi(U, V)$ is no greater that the ϕ-quantile distance of object pairs from $\mathcal{U} \times \mathcal{V} - \mathcal{P}$.

2.2 Preliminaries

ϕ-quantile Distance Computation. Given a collection S of m elements, each element s_i has a weight $w(s_i)$ where $0 < w(s_i) \leq 1$ and $\sum_{i=1}^{m} w(s_i) \leq 1$. A naive

[1] Note that our techniques developed in this paper is based on Euclidean distance; nevertheless they can be immediately extended to cover other distance metrics.

way to compute the ϕ-quantile is to firstly sort S regarding a given search key f, and then scan the sorted list to obtain the ϕ-quantile of S. Clearly, the naive algorithm runs in $O(m \log m)$. In [5], an efficient and effective partitioning technique is proposed to find an element $s \in S$ to divide S into two sub-collections S_1 and S_2 with the following properties:

1. for each $s' \in S_1$, $f(s') \leq f(s)$; and for each $s' \in S_2$, $f(s') \geq f(s)$.
2. $|S_1| \geq \frac{3}{10}m - 6$ and $|S_2| \geq \frac{3}{10}m - 6$.

Using the partitioning technique, when S is not sorted the time complexity of computing ϕ-quantile of S is linear - $O(|S|)$.

Regarding two multi-valued objects U and V, there are totally $|U| \times |V|$ instance pairs. Directly applying the partition based algorithm, computing ϕ-quantile distance between U and V takes $O(|U| \times |V|)$. In [21], instances inside one multi-valued object are indexed by an R-tree. Based on the R-tree, pruning techniques are proposed to discard instance pairs which are guaranteed not to be the ϕ-quantile of $U \times V$. In this paper, we use the pruning techniques enhanced, partition based, linear time complexity algorithm in [21] as a black box in computing ϕ-quantile distance between two multi-valued objects.

Conventional Top-k Similarity Joins. Conventional top-k similarity joins, also called closest pair queries, have been extensively studied over conventional (point) spatial databases [3], [6], [9]. The most recent technique proposes to build an index on the fly [3]. Nevertheless, this technique cannot be used to prune a group of object pairs. That is, every object has to participate in the distance computation. As the quantile distance computation between two objects is very expensive with the presence of multiple instances, in this paper, we will apply an R-tree index based top-k similarity join algorithm to facilitate the prevention of computing quantile distances between unpromising pairs of multi-valued objects. In [6], several algorithms are proposed using R-tree based indexes including exhaustive algorithm, recursive algorithm and Heap algorithm. Among all techniques, Heap algorithm demonstrates a better performance in most experiment settings. The priority query based algorithm in [9] is quite similar to Heap algorithm except that Heap algorithm performs a simple pruning before inserting an entry pair into the heap. We adopt the Heap algorithm and develop novel pruning techniques to speed up the computation. Note that our pruning techniques are general enough to be plugged into any R-tree based algorithm for computing conventional top-k similarity joins.

3 Framework

Our techniques for solving the top-k similarity join based on quantile distance follow a standard seeding-filtering-refinement framework outlined in Algorithm 1.

In the seeding phase, we choose k object pairs and compute their ϕ-quantile distances, using the techniques introduced in Section 2.2. Let λ_k be the maximal of these k ϕ-quantile distances, in the filtering phase, λ_k could be used to prune unpromising object pairs and iteratively updated if necessary. Any k object

Algorithm 1: Framework

- **Phase 1 - Seeding:** Compute the ϕ-quantile distance for each of the k chosen object pairs from $\mathcal{U} \times \mathcal{V}$.
- **Phase 2 - Filtering:** Discard unpromising object pairs from $\mathcal{U} \times \mathcal{V}$.
- **Phase 3 - Refinement:** Determine the final solution for ϕ-quantile top-k similarity join.

pairs from $\mathcal{U} \times \mathcal{V}$ could be chosen to compute the ϕ-quantile distance in the seeding phase. Apparently, similar object pairs will lead to smaller λ_k values; and hence better pruning power in the filtering phase. In our framework, to select k object pairs, we first use the mean $\mu(U)$ of the multiple instances for each multi-valued object U from the two given datasets to represent U. $\mu(U) = \sum_{i=1}^{m} w(u_i) \times u_i$ where m is the number of instances in U. Clearly $\mu(U)$ is also in the d-dimensional space. Thus the top-k similarity join is converted to join over conventional datasets where each object is a single point in the multi-dimensional space, and we could apply the existing algorithms [6] to obtain the k most similar pairs from the two (single-valued) datasets. The corresponding k multi-valued object pairs from $\mathcal{U}$ and $\mathcal{V}$ are then chosen to compute the ϕ-quantile distances. At this point, we obtain a distance threshold λ_k which will be used in the filtering phase.

Data Structures

In our techniques, we use aggregate R-trees [16] to index the local instances of each multi-valued object in $\mathcal{U} \cup \mathcal{V}$, and use two statistic information enhanced R-trees (named sR-trees) to globally index the minimum bounding boxes (MBBs) of objects in $\mathcal{U}$ and $\mathcal{V}$, respectively. The local aR-trees and global sR-tress are built to facilitate our filtering techniques.

Local aR-trees. For each multi-valued object $U \in \mathcal{U} \cup \mathcal{V}$, a *local* aR-tree [16] is built to organize its multiple instances. The aggregate information kept on each intermediate entry is the sum of weights of instances indexed by the entry. Namely, for every intermediate entry E in the local aR-tree, we record the weight of E as the sum of weights (total weights) of instances having E as an ancestor.

Global sR-trees. We maintain two R-trees on the MBBs of multiple instances of objects in $\mathcal{U}$ and $\mathcal{V}$, respectively. That is, for each object in $\mathcal{U}$, we first obtain the MBB of its multiple instances. Then we build an R-tree on these MBBs. This R-tree is called the *global* R-tree of $\mathcal{U}$. Similarly we build the *global* R-tree for $\mathcal{V}$. Note in a global R-tree, each leaf (data) entry is an MBB of an object.

Suppose an object U has m instances in the d-dimensional space, $u_1, u_2, ..., u_m$ with the weights $w(u_1), w(u_2), ..., w(u_m)$, respectively.

Definition 3 (Mean μ). *The mean of U, denoted by $\mu(U)$, is $\sum_{i=1}^{m} w(u_i) \times u_i$.*

Note that $\mu(U)$ is in the d-dimensional space. For $1 \leq i \leq d$, $\mu_i(U)$ denotes the i-th coordinate of $\mu(U)$.

Definition 4 (Variance σ^2). *For $1 \leq i \leq d$, $\sigma^2(U) = \sum_{j=1}^{m} w(u_j)(u_{j,i} - \mu_i(U))^2$ where each $u_{j,i}$ denotes the i-th coordinate value of u_j.*

In each of the leaf (data) entry of the global R-tree, besides the MBB information of each object, we also keep the above statistic information. And the global R-tree is called a statistic R-tree, denoted by sR-tree. Remind that two sR-tree are built for the multi-valued object sets $\mathcal{U}$ and $\mathcal{V}$, respectively.

4 ϕ-Quantile Top-k Similarity Join

We present our techniques for ϕ-quantile top-k similarity join for a given $\phi \in (0, 1]$ in this section. We first present novel distance, statistic and weight based pruning techniques. Then, we integrate the proposed pruning techniques into the overall join algorithm based on the Heap Algorithm in [6].

4.1 Pruning Techniques

When introducing the pruning techniques, we assume that we have an entry pair (E_U, E_V) from the join processing where E_U (E_V) is an entry from the global sR-tree of $\mathcal{U}$ $(\mathcal{V})$. E_U (E_V) could be either intermediate or leaf (data) entry. The way to access entries from the two global sR-trees will be introduced in Section 4.2.

Distance based Pruning. The first pruning rule is based on the distance between two entries in the join processing obtained from intermediate or leaf entries of two global sR-trees.

Pruning Rule 1. Let $d^{lo}(E_U, E_V)$ denote the minimum distance between the MBBs of two entries E_U and E_V. If $d^{lo}(E_U, E_V) \geq \lambda_k$, then (E_U, E_V) can be pruned, namely, all entry pairs in $E_U \times E_V$ can be pruned.

Complexity. Computing the minimum distance between two MBBs takes $O(d)$ time. The complexity of Pruning Rule 1 is constant once d is fixed.

Statistic based Pruning. The second pruning technique utilizes the statistic information kept in the global sR-tree, as introduced in Section 3. The main idea is based on the current distance threshold λ_k, to derive a value α such that the α-quantile distance between an object pair (U, V) is not smaller than λ_k. If $\alpha < \phi$, we can safely prune (U, V). We first introduce the *Cantelli's inequality* [15] which is employed in Pruning Rule 2.

Let $\delta(x, y)$ be $\frac{1}{1+\frac{x^2}{y^2}}$ if $y \neq 0$, 1 if $x = 0$ and $y = 0$, and 0 if $x \neq 0$ and $y = 0$.

Theorem 1 (Cantelli's Inequality [15]). *Suppose that t is a random variable in 1-dimensional space with mean $\mu(t)$ and variance $\sigma^2(t)$, $Prob(t - \mu(t) \geq a) \leq \delta(a, \sigma(t))$ for any $a \geq 0$, where $Prob(t - \mu(t) \geq a)$ denotes the probability of $t - \mu(t) \geq a$.*

Note that Theorem 1 extends the original Cantelli's Inequality [15] to cover the case when $\sigma = 0$ and/or $a = 0$. The following theorem is proved in [13] and provides an upper-bound for $Prob(t \leq b)$ when $b \leq \mu$.

Theorem 2. *Assume that* $0 \leq b \leq \mu(t)$. *Then,* $Prob(t \leq b) \leq \delta(\mu(t) - b, \sigma(t))$.

Proof. Let $t' = 2\mu(t) - t$. It can be immediately verified that $\sigma^2(t') = \sigma^2(t)$ and $\mu(t) = \mu(t')$. Applying Cantelli's Inequality on t', the theorem holds. □

Now we generalize the above observations into our statistic based pruning rule. As shown in Figure 3, for two object entries (U, V) stored in the leaf/data entries of global sR-tree of $\mathcal{U}$ and $\mathcal{V}$, along the i-th dimension $(1 \leq i \leq d)$, e.g., the horizontal dimension in Figure 3, we locate two lines m and n vertical to the i-th dimension and with distance λ_k between m and n. Denote U_i (V_i) as the coordinate value of U (V) along the i-th dimension. The line $U_i = m$ $(V_i = n)$ divides the MBB of U (V) into two parts, denoted as U_1 and U_2 (V_1 and V_2), as shown in Figure 3. Assume $\mu_i(U) < \mu_i(V)$. Remind that λ_k is the current distance threshold.

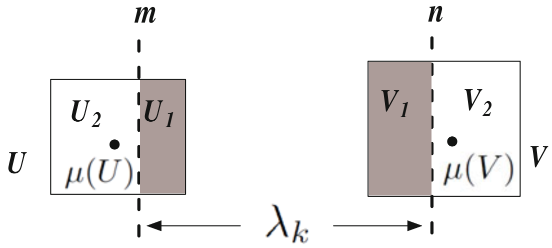


Fig. 3. Statistic based Pruning

The intuition of the statistic based pruning technique is along each dimension i, based on Theorem 2, we derive an upper bound of the sum of weights in the shaded areas of the MBBs of U_1 and V_1, respectively, denoted as $W_i^{up}(U_1)$ and $W_i^{up}(V_1)$. Clearly, we can claim that instance pairs from $U_2 \times V_2$ can not have distance smaller than λ_k. Denote the sum of weights in U_2 and V_2 as $W_i(U_2)$ and $W_i(V_2)$, respectively. Apparently, $W_i(U_2) \geq 1 - W_i^{up}(U_1)$, and $W_i(V_2) \geq 1 - W_i^{up}(V_1)$. Thus, using $W_i^{up}(U_1)$ and $W_i^{up}(V_1)$, we can identify a value α such that the α-quantile distance between U and V is not smaller than λ_k. Next we present the monotonic property of quantile distance.

Theorem 3 (Monotonicity of Quantile Distance). *Given two multi-valued objects* U *and* V, $\alpha, \phi \in (0, 1]$, *if* $\alpha < \phi$, *then* $d_\alpha(U, V) \leq d_\phi(U, V)$.

Proof. The theorem immediately holds based on the definition of quantile distance in Definition 2. □

Based on Theorem 3, once we identify the value α such that the α-quantile distance between U and V is larger than λ_k, if $\alpha < \phi$, then we can claim the ϕ-quantile distance between U and V cannot be smaller than λ_k. In this way (U, V) can be pruned based on the statistic information kept in the global sR-tree only without accessing the local aR-trees of U and V.

Pruning Rule 2. Given an object pair (U, V) $(U \in \mathcal{U}, V \in \mathcal{V})$. For a dimension i $(1 \leq i \leq d)$, without lose of generality, assume $\mu_i(U) < \mu_i(V)$. If 1 - (1 - $\delta(m - \mu_i(U), \sigma_i(U))) \times (1 - \delta(\mu_i(V) - m, \sigma_i(U))) < \phi$, (U, V) can be pruned.

Proof. For the i-th $(1 \leq i \leq d)$ dimension, based on Theorem 2, we obtain the upper bound of the sum of weight of instances in the shaded area U_1 of the MBB of U as $W_i^{up}(U_1) = Prob(U_i \geq m) \leq \delta(m - \mu_i(U), \sigma_i(U))$. Similarly we get $W_i^{up}(V_1) = Prob(V_i \leq n) \leq \delta(\mu_i(V) - n, \sigma_i(V))$. Since the instance pairs from $U_2 \times V_2$ cannot have distance smaller than λ_k, we have $\alpha \leq 1 - (1 - W_i^{up}(U_1)) \times (1 - W_i^{up}(V_1)) \leq 1 - (1 - \delta(m - \mu_i(U), \sigma_i(U))) \times (1 - \delta(\mu_i(V) - m, \sigma_i(U)))$. Together with Theorem 3, the pruning rule is correct. □

Once we obtain an object pair (U, V) from the join processing, we apply Pruning Rule 2 based on the statistic information kept in the global sR-trees before accessing the local aR-trees of U and V. If we encounter a dimension i such that 1 - (1 - $W_i^{up}(U_1)) \times (1 - W_i^{up}(V_1)) < \phi$, the pruning rule stops and the object pair (U, V) is discarded. As shown in Figure 3, after selecting line m along the i-th dimension of U, line n for V is also fixed regarding the current λ_k. We apply the equality principle in determining the position of m and n; namely, the center of m and n is the same as the center of $\mu_i(U)$ and $\mu_i(V)$. Based on Theorem 2, we obtain $W_i^{up}(U_1)$ and $W_i^{up}(V_1)$ in constant time.

Complexity. If $W_i^{up}(U_1)$ and $W_i^{up}(V_1)$ are derived based on Theorem 2, the time complexity of Pruning Rule 2 is $O(d)$.

Weight based Pruning. The following pruning rule incorporates both weight and distance information. The instances of a multi-valued object are investigated by accessing the local aR-trees. Consider an object entry pair (U, V). If (U, V) is not pruned by Pruning Rule 1 and 2, we explore the instances information of the objects by accessing their local aR-trees. We traverse the local aR-trees of two objects U and V synchronously. At level i, we *trim* object V using the current distance threshold λ_k, and retain only the entries in V with minimum distance to U not larger than λ_k. We record the entries as $\gamma_{V,i}$. Formally, $\gamma_{V,i}$ = $\{E \in L_i(V), d^{lo}(U, E) \leq \lambda_k\}$, where $L_i(V)$ denotes all remaining entries (i.e., not trimed in higher levels) in the local aR-tree of V at the i-th level. Similarly, we obtain $\gamma_{U,i}$. If the multiplication of the weights of $\gamma_{V,i}$ and $\gamma_{U,i}$ is smaller than ϕ, the object pair (U, V) can be pruned as the ϕ-quantile distance between U and V must be larger than λ_k.

Pruning Rule 3. If $\sum_{e \in \gamma_{U,i}} W(e) \times \sum_{e \in \gamma_{V,i}} W(e) < \phi$, the object pair (U, V) can be discarded.

Proof. From the definition of ϕ-quantile distance, it is immediate that if $\sum_{e \in \gamma_{U,i}} W(e) \times \sum_{e \in \gamma_{V,i}} W(e) < \phi$, then $d_\phi(U, V) > \lambda_k$. □

Example 2. As shown in Figure 4, at the i-th level, the local aR-tree of object U has two entries U_1 and U_2, local aR-tree of V also has two entries V_1 and V_2. The current threshold λ_k is as illustrated. Using λ_k, we *trim* the MBB of V and only entry V_1 has minimum distance to U smaller than λ_k; thus, $\gamma_{V,i} = \{V_1\}$. Similarly, $\gamma_{U,i} = \{U_2\}$. If $W(U_2) \times W(V_1) < \phi$, the object pair (U, V) could be pruned.

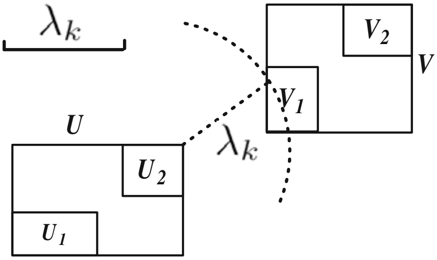


Fig. 4. Weight based Pruning

Applying Pruning Rule 3, we can avoid accessing all instance pairs of $U \times V$, and seek to stop on intermediate levels of the local aR-trees of U and V. Note the traversal of two aR-trees is in a synchronous fashion and level-by-level from the root node. If one aR-tree reaches leaf nodes first, it stays in leaf level while the other one keeps traversing till its leaf level. As a by-product, if (U, V) cannot be pruned using Pruning Rule 3, we call the ϕ-quantile distance computation algorithm in [21] with the instance pairs from $\gamma_{U,i} \times \gamma_{V,i}$ only where i is the leaf (instance) level. Clearly, the algorithm still outputs correct ϕ-quantile distance as the distance of the pruned instance pairs are larger than λ_k based on the definition of $\gamma_{U,i}$ and $\gamma_{V,i}$ for level i.

An exceptional case of Pruning Rule 3 is that we obtain an entry pair (E_U, E_V) sfrom the join processing, one is an object entry while the other is an intermediate entry. Assume E_U is the object entry of U and E_V is the intermediate entry. Pruning Rule 3 could still be applied to (U, E_V) with the following modifications: 1) We access the local aR-tree of U only and at each level i, record $\gamma_{U,i}$ as the entries in U with minimum distance to E_V not larger than λ_k; 2) if $\sum_{e \in \gamma_{U,i}} W(e) < \phi$, the entry pair (U, E_V) could be pruned. Namely, the object pair of U and any object indexed in E_V must have a ϕ-quantile distance greater than λ_k.

Complexity. Assume the average number of entries at level i of the local aR-trees of multi-valued objects is N_i, then clearly the complexity of Pruning Rule 3 is $O(N_i)$ at each level. The worst case complexity of using Pruning Rule 3 is $O(|U| \times |V|)$, namely no entries are pruned at intermediate entries and we need to access all instance pairs. However, in practice, as shown in Section 5, Pruning Rule 3 is very effective and saves CPU costs significantly. Note that in Pruning

Rule 3 we trim the entries at each level of local aR-trees of U and V using λ_k instead of considering the combination of all pairs of entries at each level. This is because trim based pruning is more efficient compared with combining all pairs (time complexity $O(N_i^2)$) and also trim based pruning is very effective in practice.

4.2 Overall Join Algorithm

The join algorithm used in this paper is adopted from the Heap Algorithm in [6] as it is both efficient and easy to implement in real applications. We adjust the algorithm to deal with multi-valued objects. Given $\phi \in (0, 1]$, two multi-valued objects sets $\mathcal{U}$ and $\mathcal{V}$, Algorithm 2 illustrates the top-k similarity join processing. A minheap H is maintained according to the minimum distance between two entry pairs of the two global R-trees $R_{\mathcal{U}}$ and $R_{\mathcal{V}}$ indexing $\mathcal{U}$ and $\mathcal{V}$, respectively. H is initialized with the pair of root nodes of $R_{\mathcal{U}}$ and $R_{\mathcal{V}}$.

Algorithm 2: Top-k Similarity Join Processing

```
   Input  : $R_{\mathcal{U}}$, $R_{\mathcal{V}}$, $k$, $\phi$
   Output : $k$ object pairs from $\mathcal{U} \times \mathcal{V}$ with smallest $\phi$-quantile distances
1  $H$ = (root($R_{\mathcal{U}}$), root($R_{\mathcal{U}}$)) if not PRUNED1(root($R_{\mathcal{U}}$), root($R_{\mathcal{U}}$));
2  while $H$ is not empty do
3      $(E_U, E_V) = H$.top();
4      $H$. pop();
5      if $E_U$ and $E_V$ are both intermediate entries then
6          for each children pair $(C_{E_U}, C_{E_V})$ from $E_U \times E_V$ do
7              if not PRUNED1($C_{E_U}, C_{E_V}$) then
8                  insert $(C_{E_U}, C_{E_V})$ into $H$;
9      else if one of $E_U$ and $E_V$ is an object entry then
10         if not PRUNED1($E_U, E_V$) and not PRUNED3($E_U, E_V$) then
11             Lines 6 - 8;
12     else                              /* both $E_U$ and $E_V$ are object entries */
13         if not PRUNED1($E_U, E_V$) and not PRUNED2($E_U, E_V$) AND not
           PRUNED3($E_U, E_V$) then
14             Compute $\phi$-quantile distance between $E_U$ and $E_V$;
15             if $d_\phi(E_U, E_V) < \lambda_k$ then
16                 Update $\lambda_k$ and current $k$ most similar pairs;
17
```

The algorithm differentiates three cases based on whether the entries are object entries or not. If both are intermediate entries (Line **5**), we expand all the children pairs and insert into heap H the pairs which survive from Pruning Rule 1 (Line **7**). If one of the entries is an intermediate entry and the other is an object entry (Line **9**), Pruning Rule 1 and 3 will be applied first (Line **10**) before expanding the children pairs. We apply all 3 Pruning Rules object pairs (Line **13**), and if survived, the ϕ-quantile distance is computed; the top-k results and λ_k are updated if necessary. Note that even from the root node pair we only insert entry pairs into H if they are not pruned by Pruning Rule 1, it is still necessary to check Pruning Rule 1 (Line **9** and Line **13**) since the distance threshold λ_k dynamically changes.

Correctness. Based on the correctness of the 3 pruning rules, it can be immediately shown that Algorithm 2 is correct.

Discussions. The techniques proposed in this paper could be immediately extended to support self-join (i.e., we compute top-k similar pairs from one data set $\mathcal{U}$) and threshold base similarity join over multi-valued objects. We omit the details due to space limits.

5 Experiment

We report a thorough performance evaluation on the efficiency and effectiveness of our algorithms. In particular, we implement and evaluate the following techniques.

Top-k Join: Techniques presented in Section 4 to compute top-k similarity join based on ϕ-quantile distance ($\phi \in (0, 1]$), with all 3 pruning techniques.
P12: Techniques in Section 4 but using Pruning Rule 1 and 2 only (i.e., distance and statistic based pruning).
P1: Techniques in Section 4 but using Pruning Rule 1 only (i.e., distance based pruning).
P0: Techniques in Section 4 but without any pruning rule.
KNN: Baseline algorithm by using KNN processing over multi-valued objects in [21]. For each object $U \in \mathcal{U}$, we compute its KNN in $\mathcal{V}$, and then select k most similar pairs based on the union of KNN results.

All algorithms are implemented in C++ and compiled by GNU GCC. Experiments are conducted on PCs with Intel Xeon 2.4GHz dual CPU and 4G memory under Debian Linux. Our experiments are conducted on both real and synthetic datasets.

Real dataset is extracted from NBA players' game-by-game statistics containing 339,721 records of 1,313 players (http://www.nba.com). Each player is treated as a multi-valued object where the statistics (score, assistance, rebound) of a player per game is treated as an instance with the equal weight (normalized).

Synthetic datasets are generated using the methodologies in [1] regarding the following parameters. Dimensionality d varies from 2 to 5 with default value 3. Data domain in each dimension is $[0, 1]$. Average number n of objects in each dataset varies from $5k$ to $15k$ with default value $5k$. Number of instances per object follows a uniform distribution in [1, m] where m varies from 100 to 800 with the default value 200. The value K varies among 5 to 25 with default value 10. The average length of object MBBs varies from 0.01 to 0.05 with default value 0.01. With the default setting, the total number of instances in a dataset is about $500k$.

Generating $\mathcal{U}$ and $\mathcal{V}$. In a (real or synthetic) dataset, each object is drawn to $\mathcal{U}$ or $\mathcal{V}$ with equal probability (i.e., probability of $\frac{1}{2}$).

5.1 Overall Performance

Figure 5 reports the results of the evaluation on processing time of Top-k Join, P12, P1, P0, and KNN over real and synthetic data. KNN is very slow and cannot terminate when the dataset size is large, so in the synthetic dataset, we set dataset size to 1k and other parameters take the default values. As shown, each pruning rule is very effective and reduce the processing time significantly. Our techniques are much more efficient compared to a naive application of KNN techniques to processing top-k similarity joins. As P0 and KNN are very inefficient, we omit their performance evaluation in the following sections.

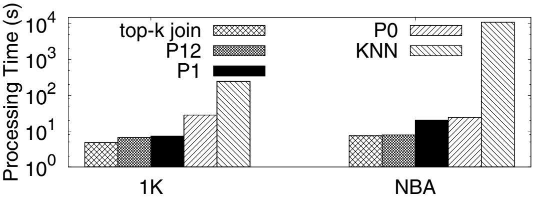


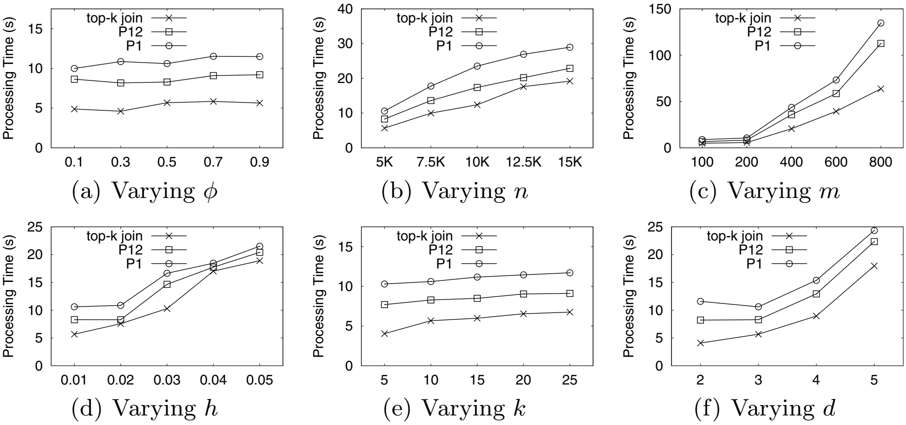
Fig. 5. Overall Performance

5.2 Evaluating Impacts by Different Settings

In this subsection, we study the scalability of our algorithms regarding different ϕ-values, number of objects in one dataset(n), number of instances (m), length of MBB edges (h), k and the dimensionality d in Figure 6. While our techniques are not very sensitive to ϕ-values and k, the processing time increases with the increase of number of objects, instance number, MBB edge length, and dimensionality. Clearly, the dataset size increases with objects and instances number thus the join processing becomes more expensive. Larger MBB edge length makes it difficult to prune object pairs as there is larger overlap in their MBBs. The results also demonstrate that each pruning rule is very effective and significantly reduces the processing time.

5.3 I/O Costs

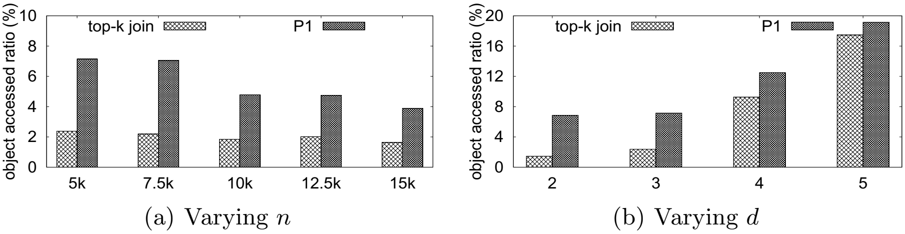
We report the ratio of objects accessed in Figure 7. Since Pruning Rule 3 mainly works on two object entries and load both two objects, its effect is on saving



(a) Varying ϕ (b) Varying n (c) Varying m

(d) Varying h (e) Varying k (f) Varying d

Fig. 6. Varying Parameters



(a) Varying n (b) Varying d

Fig. 7. Object Accessed Ratio

CPU time but not I/O. Thus, we report the performance of Top-k Join and P1 only. As shown in the figure, our techniques could achieve dramatic saving on the total objects loaded. Compared to P1, the saving from using Pruning Rule 2 is also significant.

6 Related Work

Conventional join queries over two multi-dimensional datasets are fundamental in data analysis and information retrieval. Most existing techniques for join queries have been developed based on popular spatial access methods such as R-trees. For threshold based joins, there are three main stream spatial join algorithms using R*-tree [8]. They are the depth-first-join (DFJ) algorithm [2], the breadth-first-join (BFJ) algorithm [10], and transformation-view-join (TVJ) algorithm [12]. Techniques for top-k spatial/similarity queries are studied in [6], [9]. Various algorithms, such as exhaustive algorithm, recursive algorithm, Heap algorithm, and priority queue based algorithms are proposed. Many variation of join queries over multi-dimensional space have been studied in different contexts, including road networks [18] and moving objects [20].

Spatial queries such as nearest neighbor queries over fuzzy objects have been recently studied [22]. Fuzzy objects possess similar semantics as uncertain objects (e.g., instances are mutually exclusive). The techniques in [22] are not applicable to the problem studied in our paper due to the different semantics as well as inherent difference in query types.

Join queries over uncertain objects are inherently different than conventional joins where each uncertain object takes a set of mutually exclusive instances/points in a multi-dimensional space. It is extensively studied in [4], [11], [14]. Note that all instances in a multi-valued object exist simultaneously instead of mutually exclusive in an uncertain object. Due to such inherent differences in semantics, join techniques over uncertain objects cannot be directly applied to similarity joins over multi-valued objects.

7 Conclusion

We study the problem of top-k similarity join over multi-valued objects. The distance/similarity between two multi-valued objects is measured using quantile based distance to capture the relative instance distribution. A filtering-refinement framework is developed, along with novel, efficient and efficient distance, statistic and weight based pruning techniques. Comprehensive experimental study over both real and synthetic datasets demonstrates the efficiency and scalability of our techniques.

Acknowledgement. Wenjie Zhang is supported by ARC DP120104168 and ARC DE120102144. Ying Zhang is supported by ARC DP110104880 and UNSW ECR grant PSE1799. Xuemin Lin is supported by ARC DP0987557, ARC DP110102937, ARC DP120104168, NSFC61021004 and a Google Research Award.

References

1. Borzsonyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE 2001 (2001)
2. Brinkhoff, T., Kriegel, H.-P., Seeger, B.: Efficient processing of spatial joins using r-trees. In: SIGMOD 1993 (1993)
3. Cheema, M.A., Lin, X., Wang, H., Wang, J., Zhang, W.: A unified approach for computing top-k pairs in multidimensional space. In: ICDE 2011 (2011)
4. Cheng, R., Singh, S., Prabhakar, S., Shah, R., Vitter, J.S., Xia, Y.: Efficient join processing over uncertain data. In: CIKM 2006 (2006)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. Medians and order statistics, 2nd edn., ch. 9. The MIT Press
6. Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: Closest pair queries in spatial databases. In: SIGMOD 2000 (2000)
7. Elmasri, R., Navathe, S.: Fundamentals of database systems, 6th edn. (2011)
8. Han, W.-S., Kim, J., Lee, B.S., Tao, Y., Rantzau, R., Markl, V.: Cost-based predictive spatiotemporal join. In: TKDE 2009 (2009)

9. Hjaltason, G., Samet, H.: Incremental distance join algorithms for spatial databases. In: SIGMOD 1998 (1998)
10. Huang, Y.-W., Ning, J., Rundensteiner, E.A.: Spatial joins using r-trees: Breadth-first traversal with global optimizations. In: VLDB 1997 (1997)
11. Kriegel, H.-P., Kunath, P., Pfeifle, M., Renz, M.: Probabilistic Similarity Search on Uncertain Data. In: Li Lee, M., Tan, K.-L., Wuwongse, V. (eds.) DASFAA 2006. LNCS, vol. 3882, pp. 295–309. Springer, Heidelberg (2006)
12. Lee, M.-J., Whang, K.-Y., Han, W.-S., Transform-space, S.I.-Y.: view: Performing spatial join in the transform space using original-space indexes. In: TKDE 2006 (2006)
13. Lin, X., Zhang, Y., Zhang, W., Cheema, M.A.: Stochastic skyline operator. In: ICDE 2011 (2011)
14. Ljosa, V., Singh, A.K.: Top-k spatial join of probabilistic objects. In: ICDE 2008 (2008)
15. Meester, R.: A Natural Introduction to Probability Theory (2004)
16. Papadias, D., Kalnis, P., Zhang, J., Tao, Y.: Efficient OLAP Operations in Spatial Data Warehouses. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) SSTD 2001. LNCS, vol. 2121, pp. 443–459. Springer, Heidelberg (2001)
17. Rigaux, P., Scholl, M., Voisard, A.: Spatial databases: With applications to gis (2001)
18. Sankaranarayanan, J., Alborzi, H., Samet, H.: Distance join queries on spatial networks. In: GIS 2006 (2006)
19. Yiu, M.L., Mamoulis, N., Tao, Y.: Efficient Quantile Retrieval on Multi-Dimensional Data. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Böhm, K., Kemper, A., Grust, T., Böhm, C. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 167–185. Springer, Heidelberg (2006)
20. Zhang, R., Lin, D., Ramamohanarao, K., Bertino, E.: Continuous intersection joins over moving objects. In: ICDE 2008 (2008)
21. Zhang, W., Lin, X., Cheema, M.A., Zhang, Y., Wang, W.: Quantile-based knn over multi-valued objects. In: ICDE 2010 (2010)
22. Zheng, K., Fung, P., Zhou, X.: K nearest neighbor search for fuzzy objects. In: SIGMOD 2010 (2010)

Indexing Network Voronoi Diagrams[*]

Ugur Demiryurek and Cyrus Shahabi

University of Southern California,
Department of Computer Science
Los Angeles, CA 90089-0781
{demiryur,shahabi}@usc.edu

Abstract. The Network Voronoi diagram and its variants have been extensively used in the context of numerous applications in road networks, particularly to efficiently evaluate various spatial proximity queries such as k nearest neighbor (kNN), reverse kNN, and closest pair. Although the existing approaches successfully utilize the network Voronoi diagram as a way to partition the space for their specific problems, there is little emphasis on how to efficiently find and access the network Voronoi cell *containing* a particular point or edge of the network. In this paper, we study the index structures on network Voronoi diagrams that enable exact and fast response to contain query in road networks. We show that existing index structures, treating a network Voronoi cell as a simple polygon, may yield inaccurate results due to the network topology, and fail to scale to large networks with numerous Voronoi generators. With our method, termed Voronoi-Quad-tree (or VQ-tree for short), we use Quad-tree to index network Voronoi diagrams to address both of these shortcomings. We demonstrate the efficiency of VQ-tree via experimental evaluations with real-world datasets consisting of a variety of large road networks with numerous data objects.

1 Introduction

The latest developments in wireless technologies as well as the widespread use of GPS-enabled mobile devices have led to the recent prevalence of location-based services. An important class of location based queries consists of proximity queries such as k Nearest Neighbor(kNN) query [15,32,21,6,7] and its variations, e.g., Reverse k Nearest Neighbor (RkNN) [23,29], k Aggregate Nearest Neighbor (kANN) [28]. The proximity queries in general search for data objects that minimize a distance-based function with reference to one or more query objects.

[*] This research has been funded in part by NSF grants IIS-0238560 (PECASE), IIS-0534761,IIS-0742811 and CNS-0831505 (CyberTrust), and in part from the METRANS Transportation Center, under grants from USDOT and Caltrans.Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We thank Prof. Ulrich Neumann for his insightful discussions and comments.

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 526–543, 2012.
© Springer-Verlag Berlin Heidelberg 2012

With proximity queries, potentially the distance between the query point and every object in the database (e.g., all the points-of-interest) must be computed in order to find the closest (or the k closest) object(s) to the query point. Hence, the main research focus has been on indexing the objects to avoid the exhaustive search. Earlier studies assumed Euclidean distance as the distance function and hence indexed the objects in Euclidean space (e.g., [32,30,21,24]) using R-tree [4] like index structures. With the advent of online mapping systems such as Google Maps and Mapquest and the availability of accurate nation-wide road network data, the proximity queries have been extended from Euclidean space to the road network space as natural artifact. The challenge in processing proximity queries on road networks is that the computation of the distance function is complex and hence the indexing techniques incorporated some sort of pre-computation of distances (in network) into their structures. One such approach is based on using network Voronoi diagrams [12].

A network Voronoi diagram is a specialization of a Voronoi diagram in which the locations of objects are restricted to the network edges and the distance between objects is defined as the length of the shortest network distance (e.g., shortest path or shortest time), instead of the Euclidean distance. Any network node located in a Voronoi cell has a shortest path to its corresponding Voronoi generator that is always shorter than that to any other Voronoi generator. A large number of studies adopted network Voronoi diagrams [12] to evaluate variety of proximity queries on road networks (e.g., [7,11,13,27,17]). For example, in [13] Okabe et al. introduced six different types of network Voronoi diagrams (each corresponds to very important real-world applications) whose generators are based on points, sets of points, lines and polygons, and whose distances are given by inward/outward distances, and additively/multiplicatively weighted shortest path distances.

Given a query point q and network Voronoi diagram (NVD), the first step in answering any proximity query is to locate the network Voronoi cell $NVC(p_i)$ that contains q (the generator p_i of $NVC(p_i)$ is the nearest neighbor of q). We refer to this operation as $contain(q)$ in the rest of the paper. Considering the large size of the underlying space (e.g., a continental size road network) with numerous data objects as well as the online nature of the queries that requires fast response-time, an index structure is necessary to efficiently access the portion of NVD associated with q. Although the existing approaches successfully used network Voronoi diagrams as a pre-computation approach for partitioning the network space, they overlooked the indexing techniques that enable efficient evaluation of $contain(q)$. Currently, indexing network Voronoi diagram with R-tree (referred as Voronoi R-tree or VR-tree for short) is the only known method for locating the network Voronoi cell that contains a particular point or edge of the network. VR-tree is first proposed in [7] and later used in many other approaches based on NVD (e.g., [11,27,17]).

In this paper, we show that VR-tree has two main problems. First, VR-tree may yield inaccurate results due to the way the Voronoi cells are formed in network space, i.e., although a NVD is generated based on the network

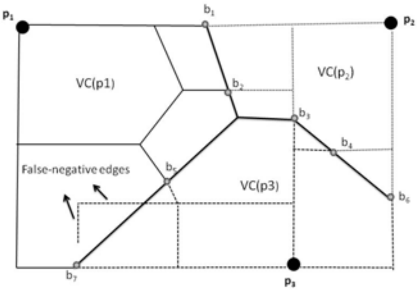


Fig. 1. Network Voronoi Diagram

distance metric, its Voronoi cells are created and indexed as regular polygons in Euclidean space. This inconsistency may result in a network edge belonging to a cell $NVC(p_i)$, to be classified as a member of the cell $NVC(p_j)$ because due to the network topology, the edge falls inside the polygon of $NVC(p_j)$ even though its network distance is closer to the generator of $NVC(p_i)$. For example, Figure 1 depicts the network Voronoi diagram of a hypothetical road network where each line style corresponds to network Voronoi cells of the generators p_1, p_2 and p_3. With VR-tree the network Voronoi cells are formed by connecting the border points (i.e., $\{b_1, b_2, ..., b_7\}$)[1] and bounded by straight line segments (i.e., bold lines in the Figure). As shown, the edges marked by *false-negative* are included in the Voronoi cell of p_1 $NVC(p_1)$, however the network distance from any point on the false-negative edges to p_3 is shorter than that to p_1.

Second, VR-tree is inefficient because of the non-disjoint partitioning of the space. Specifically, VR-tree splits the network space with hierarchically nested and largely overlapping minimum bounding rectangles (MBR) created around network Voronoi cells. The overhead of executing $contain(q)$ query is prohibitively high particularly in large networks with a dense (but perhaps large) set of data objects. This is because VR-tree has to redundantly visit the parent node(s) of the overlapping MBRs (aka, backtracking problem) in the index structure.

To address both of the aforementioned drawbacks, we propose a new indexing approach for network Voronoi diagrams based on region Quad-tree [18], termed *Voronoi-Quad-tree* or VQ-tree for short. VQ-tree, unlike VR-tree that approximates network Voronoi cells using regular polygons in the Euclidean space, enables exact representation of the network Voronoi cells based on quad-tree blocks in the network space, and hence always yields correct results. VQ-tree does not suffer from the backtracking problem of VR-tree. This is because VQ-tree enables disjoint decomposition of the network space and encodes each of the quad-tree blocks to indicate the identity of the network Voronoi cell of which it is a member. Thus, once the quad-tree block containing q is located, VQ-tree immediately identifies the nearest Voronoi generator based on the encoded value

[1] We discuss the network Voronoi diagram generation in Section 4.1.

of that block. Our experiments with real-world datasets show that the ratio of false-negative edges is %16 on average with respect to the total number of edges in the network and VQ-tree outperforms VR-tree with 12 times improved response time (see Section 5).

The remainder of this paper is organized as follows. In Section 2, we review the related work about proximity queries in spatial networks. In Section 3, we overview Network Voronoi diagrams and it's properties. In Section 4, we establish the theoretical foundation of the proposed solution for indexing Network Voronoi diagrams for efficient and accurate processing of proximity queries in spatial networks. In Section 5, we present the results of our experiments with a variety of spatial networks with large number of query and data objects. Finally, in Section 6 we conclude and discuss our future work.

2 Related Work

The most widely studied class of proximity queries consists of k nearest-neighbor (kNN) and its variations. The research on kNN query processing can be categorized into two main areas, namely, Euclidean space and road networks. In the past, numerous algorithms (e.g., [32,30,21,24,9]) have been proposed to solve kNN problem in the Euclidean space. All of these approaches are applicable to the spaces where the distance between objects is only a function of their spatial attributes (e.g., Euclidean distance). In network spaces, however, the query and data objects are located in predefined network segments, where the distance between a pair of objects is defined as the length of the shortest path connecting them.

The challenge with processing kNN queries in road-network space is that the computation of the distance function (e.g., shortest path) is complex. Therefore, to enable efficient evaluation of kNN queries in road networks, the research in this area largely focused on techniques which utilize precomputed network distances and/or partial results. One common example of such techniques is the network Voronoi diagrams. Kolahdouzan and Shahabi proposed first network Voronoi based kNN search technique, termed VN3 [7,8]. They retrieve the kNN of a query point q based on precomputed first-order network Voronoi diagram. Specifically, they first find the network Voronoi cell that contains q and then, to find k-1 nearest neighbors, search the adjacent Voronoi polygons iteratively. With their approach, they indexed the Voronoi cells with R-tree (i.e., VR-tree) to reduce the $contain(q)$ query to a point location problem in the Euclidean space. In [14], Papadias et al. introduced Incremental Network Expansion (INE) and Incremental Euclidean Restriction (IER) methods to support kNN queries in spatial networks. While INE is an adaption of the Dijkstra algorithm, IER exploits the Euclidean restriction principle in which the results are first computed in Euclidean space and then refined by using the network distance. Several other kNN algorithms are proposed based on the improved (precomputation) version of INE [1,25,5]. In [19], Samet et al. proposed *shortest path quadtree* algorithm for efficient evaluation of both shortest path and kNN queries in road networks.

VQ-tree is mainly different than the shortest path quadtree for the following reason. With SPQ-tree, N region quad-trees are created, one for each vertex of a road network (with N vertices), where each quad-tree(SPQ-tree) represents the adjacency list of its corresponding vertex as regions. However, VQ-tree is a single quad-tree created for the entire road network with each of its encoded quad-blocks corresponding to one network Voronoi cell. In [13], Okabe et al. introduced a variety of network Voronoi diagrams where they assumed Voronoi generators as points, sets of points, lines and polygons, and network distances as inward/outward, and additively/multiplicatively weighted shortest path distances. Although they proposed very useful network Voronoi diagram based solutions to real-world road network problems, they did not focus on indexing techniques that efficiently find and access the network Voronoi cells in large scale road networks. In [11], Nutanong et al. proposed a technique called local network Voronoi diagram (LNVD) to continuously monitor kNN queries in road networks. With their approach, instead of creating NVD that covers the entire road network, they construct a network Voronoi diagram for a subspace around the query point. In different studies Zhao, Xuan, Taniar and Safar et al. utilized network Voronoi diagrams to evaluate different types of proximity queries including group kNN [16], mulitple kNN [31], reverse kNN [22], and range [26] queries in road networks. With all these studies, VR-tree is used to index the network Voronoi cells. However, as we mentioned VR-tree may return false results and inefficient in large networks with numerous data objects.

3 Background

In this section, we review the principles of Euclidean and Network Voronoi diagrams. We first introduce 2-dimensional Euclidean space Voronoi diagrams and describe the properties of Voronoi diagrams. We then explain the network Voronoi diagram. We refer readers to [12] for a comprehensive discussion of Euclidean and network Voronoi diagrams.

3.1 Voronoi Diagrams

Let $P : \{p_1, p_2, .., p_n\}$ be a set of n distinct sites (i.e., generator points) distributed in the Euclidean space. These generator points can be considered any spatial type of objects (e.g., gas station, restaurant). We define the *Voronoi diagram* of P as the subdivision of the space into n cells, one for each site in P, with the property that a point q lies in the cell corresponding to a site p_i if and only if $distance(q, p_i) < distance(q, p_j)$ for each $p_j \in P$ with $j \neq i$. Figure 2 shows the ordinary Voronoi diagram of eight points where the distance metric is Euclidean.

We refer to the region containing the point p_i as its Voronoi cell $VC(p_i)$ or Voronoi polygon (see $VC(p_4)$ in the Figure). In Euclidean space, $VC(p_i)$ is a convex polygon. Each edge of $VC(p_i)$ is a segment of the perpendicular bisector of the line segment connecting p to another point of the set P. We call each of

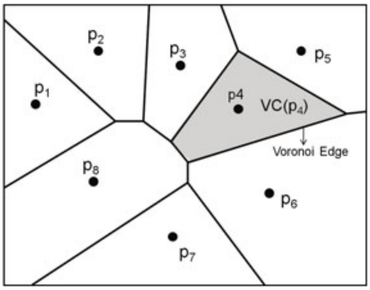


Fig. 2. Voronoi diagram in Euclidean space

these edges a Voronoi edge. The Voronoi cells that have common edges are called *adjacent cells* and their generators are called *adjacent generators*. The Voronoi cells are collectively exhaustive and mutually exclusive except their boundaries (i.e., Voronoi edges). We define the Voronoi cell and Voronoi diagram as follows.

Definition 1. *Consider $P : \{p_1, p_2, .., p_n\}$ where $2 \leq n$ and $p_i \neq p_j$ for $i \neq j$, $i, j \in I_n = 1, ...n$. The region given by $VC(p_i) = p|d(p, p_i) \leq (p, p_j)$ where $d(p, p_i)$ is the minimum Euclidean distance between p and p_i is called the Voronoi Cell (VC) associated with p_i.*

Definition 2. *The set of Voronoi cells given by $VD(P) = \{VC(p_1), ..., VC(p_n)\}$ is called the Voronoi Diagram (VD) generated by P.*

3.2 Network Voronoi Diagrams

With network Voronoi diagrams (NVD), the VD described above is generalized by replacing the Euclidean space with a spatial network (e.g., road network), hence the distance with the network distance (e.g., shortest-path) between the objects.

Definition 3. *A road network is represented as a directional weighted graph $G(N, E)$, where N is a set of nodes representing intersections and terminal points, and E ($E \subseteq N \times N$) is a set of edges representing the network edges each connecting two nodes. Each edge e is denoted as $e(n_i, n_j)$ where n_i and n_j are starting and ending nodes, respectively.*

In this study, we consider planar graph where edges intersect only at their endpoints. We assume that Voronoi generators are located on the network segments as the graph nodes. Each edge connecting nodes p_i, p_j stores the network distance $d_N(p_i, p_j)$. For nodes that are not directly connected, $d_N(p_i, p_j)$ is the length of the shortest path from p_i to p_j.

Given a weighted graph $G(N, E)$ consisting of a set of nodes $N = \{p_1, ...p_n, p_{n+1}, ..p_o\}$ where the first n nodes represent the Voronoi generators and a set of edges $E = \{e_1, ...e_k\}$ that connects the nodes, we define the set *dominance region* and *border points* as follows,

Definition 4. *The dominance region of* p_i *over* p_j $Dom(p_i, p_j) = \{p | p \in \bigsqcup_{o=1}^{k} e_o, d_N(p, p_i) \leq d_N(p, p_j)\}$ *represents all points in all edges in* E *that are closer (or equal distance) to* p_i *than* p_j*.*

Definition 5. *The border points between* p_i *and* p_j $b(p_i, p_j) = \{p | p \in \bigsqcup_{o=1}^{k} e_o, d_N(p, p_i) = d_N(p, p_j)\}$ *represent all points in all edges that are equally distanced from* p_i *and* p_j*.*

Definition 6. *Based on the above definitions, the Voronoi edge set* V_{edge} *of* p_i *as* $V_{edge}(p_i) = \bigsqcup_{j \in I_n \setminus \{i\}} Dom(p_i, p_j)$ *represents all the points in all edges in* E *that are closer to* p_i *than any other generator point in* N*. Consequently, we define network Voronoi diagram* $NVD(P)$ *w.r.t set of points* P *as* $NVD(P) = \{V_{edge}(p_1), ..., V_{edge}(p_n)\}$*.*

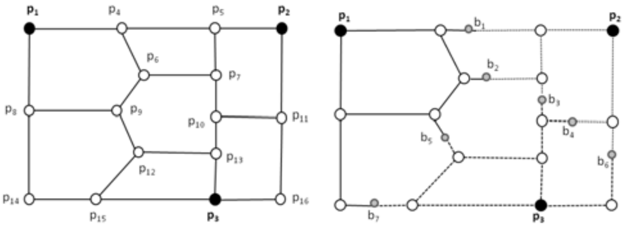
Similar to VD described in Section 3.1, the elements of NVD are mutually exclusive and collectively exhaustive.

4 Indexing Network Voronoi Diagrams

In this section, we will first explain how to construct a network Voronoi diagram in road networks and then discuss two different index structures, namely the Voronoi R-tree and Voronoi Quad-tree that efficiently identifies the subdivision of the network space that contains a particular query point or network edge.

4.1 Network Voronoi Diagram Construction

The network Voronoi diagrams can be constructed using parallel Dijkstra algorithm [2] with the Voronoi generators as multiple sources. Specifically, one can expand shortest path trees from each Voronoi generator simultaneously and stop the expansions when the shortest path trees meet.



(a) Road Network (b) Network Voronoi Diagram

Fig. 3. A Road network and network Voronoi diagram

Figure 3 shows an example of road network and the corresponding network Voronoi diagram. Figure 3a depicts the original weighted graph $G(N, E)$ which consists of $N = \{p_1, p_2, p_3, p_4, ...p_{16}\}$ nodes where p_1, p_2, and p_3 are the Voronoi generators (i.e., data objects such as restaurants, hotels) and p_4 to p_{16} are the intersections on a road network that are interconnected by a set of edges. Figure 3b shows the NVD of the road network where each line style corresponds to the shortest path tree based on the generators $\{p_1, p_2,p_3\}$. Each shortest path tree composes a network Voronoi cell and some edges (e.g., $e(p_4, p_5)$) can be partially contained in different network Voronoi cells. The border points b_1 to b_7 are the nodes where the shortest path trees meet as a result of the parallel Dijkstra algorithm. The border points between any two generator p_i and p_j are equally distanced from p_i and p_j. Figure 4 shows a real network Voronoi diagram with respect to 50 data objects in Los Angeles road network. Each network node marked with a different color corresponds to a network Voronoi cell.

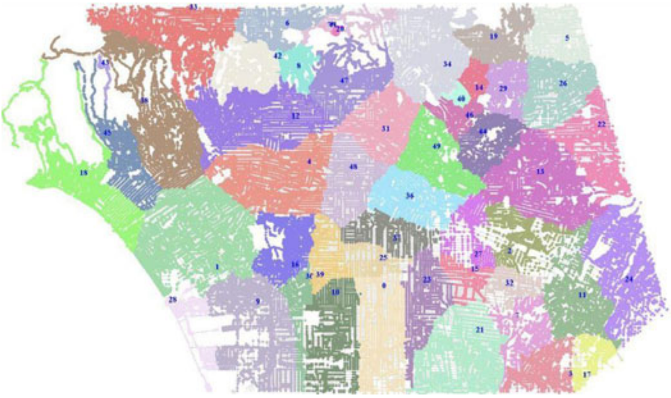


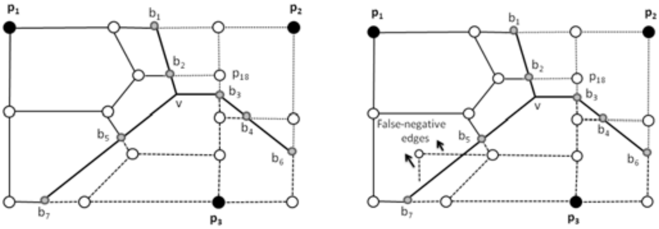
Fig. 4. Network Voronoi diagram with $P = \{p_1, ..., p_{50}\}$ in Los Angeles road network

4.2 Index Generation on Network Voronoi Diagram

As we discussed, to answer any proximity query with respect to a query point q, one first needs to find the Voronoi cell that contains q. There remains a basic question concerning how to efficiently access the portion of the NVD associated with a particular query point q. This can be achieved by utilizing a spatial index structure that is generated on Voronoi cells. Below, we discuss two types of spatial index structures that can be used to index NVCs, namely, the Voronoi R-tree(VR-tree) and Voronoi Quad-tree (VQ-tree).

4.2.1 The Voronoi R-Tree (VR-tree)

VR-tree is first introduced in [7] where NVD is used to evaluate kNN queries in road networks. VR-tree is based on the R-tree [4] that splits the network space with hierarchically nested Minimum Bound Rectangels (MBR) generated around network Voronoi cells. Given the location of a query point q, a $contain(q)$ query



(a) NVC in VR-tree (b) False-negative edges

Fig. 5. Network Voronoi cell construction in VR-tree

invoked on VR-tree starts from the root node and iteratively checks the MBRs (of NVCs) with respect to a q to decide whether or not to further search the child nodes.

VR-tree has two main shortcomings. First, VR-tree may yield inaccurate results for a *contain*(q) query. This is because VR-tree makes the simplifying assumption that although the NVD is computed based on the network distance metric, its NVCs are treated as regular polygons (by connecting border points of NVCs) and indexed using R-tree that is designed for the Euclidean distance metric. However, such approach may cause misclassification of the network edges (i.e., false-negative edges) in the network Voronoi cells, and hence inaccurate results. Specifically, a network edge belonging to a network Voronoi cell of p_i $NVC(p_i)$ may be classified as a member of another network Voronoi cell $NVC(p_j)$. For instance, continuing with our running example in Figure 3, Figure 5(a) shows how adjacent border points are connected to each other: if two adjacent border points are between two similar generators (e.g., b_5 and b_7 are between p_1 and p_3), they can be connected with an arbitrary line. Three or more adjacent border points (e.g., b_2, b_3 and b_5) can be connected to each other through an arbitrary auxiliary point (e.g., v in the figure). As a result, similar to its Euclidean counterpart, the NVCs are represented with polygons in the network space. However, to illustrate why VR-tree may fail to yield correct results, consider Figure 5(b) where we introduce two new edges (as an extension of p_{12}) to the road network. As shown, although the new edges (marked by false-negative edges in the Figure) are included inside the Voronoi cell of p_1, the network distance from any point on the false-negative edges to p_3 is shorter than that to p_1. Thus, with VR-tree, when q is located on false-negative edges, a *contain*(q) will return incorrect Voronoi generator as the NN. With our example we only show one particular case that can happen in real-world road networks. Arguably, it is possible to increase the number of such examples under different road network topologies. Figure 6 depicts the NVC of a particular data object in Los Angeles road network where border nodes and false-negative edges are marked by light blue and red color, respectively.

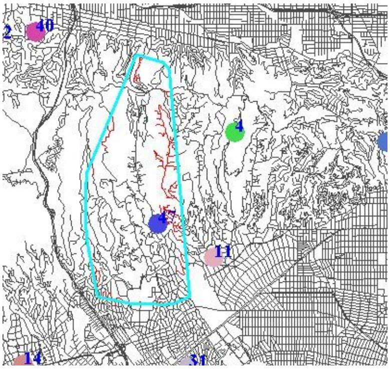


Fig. 6. False-negative edges of a NVC in Los Angeles road network

One naive solution to the inaccuracy problem of VR-tree is to perform an additional refinement step. Specifically, one can maintain false-negative edges (along with their corresponding Voronoi generators) in a separate index structure and, for each *contain*(q) query, check q against this index structure. If q is located in any of the false-negative edges, the corresponding Voronoi generator is returned as the nearest neighbor. Otherwise, VR-tree continues the search based on MBRs of the Voronoi cells as explained above.

Second, VR-tree is inefficient due to non-disjoint partitioning of the space. Specifically, with VR-tree the hierarchy of NVCs is enforced by minimum bounding rectangles created around network Voronoi cells. Depending on the different topologies of the road network and the distribution of the objects on the network segments, the overlapping areas of MBRs of network Voronoi cells may be quite large, and hence significant computation overhead in traversing R-tree for *contain*(q) query. For example, Figure 7 illustrates the MBRs of network Voronoi cells in Figure 4. For the sake of clarity, we do not include the Voronoi cells in the picture. As shown, the MBRs around network Voronoi cells result in a *non-disjoint decomposition* of the underlying space which means that the location occupied by a Voronoi cell may be contained in several bounding boxes. This degrades the search performance in VR-tree because of the backtracking [4] problem, i.e., the parent node(s) of the overlapping MBRs have to be accessed repeatedly in order to search the child nodes that contain q. Thus, with VR-tree the amount of work often depends on the overlapping areas of MBRs. We also implemented VR-tree with R+ tree [20] to reduce the impact of overlapping areas. However, we observe that the performance of VR+ tree is still less as compared to VQ-tree (see Section 5.2.4).

4.2.2 The Voronoi Quad-Tree (VQ-tree)

The alternative to VR-tree is to index network Voronoi cells using Quad-tree [18,3], termed Voronoi Quad-tree (VQ-tree), that enables disjoint decomposition of the underlying space. The main observation behind VQ-tree is that each color coded area in Figure 4 is a spatially contiguous region in the network space.

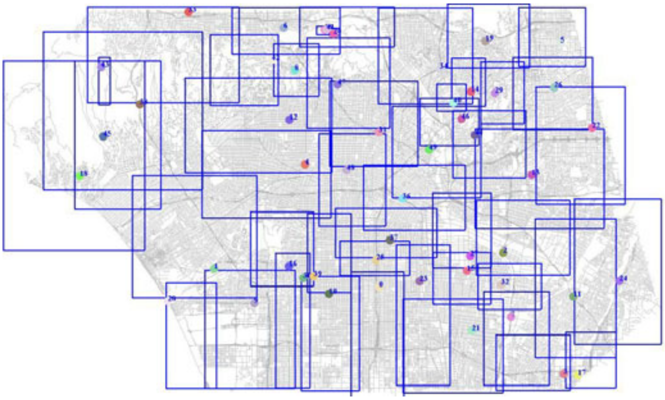


Fig. 7. Minimum bounding rectangles on network Voronoi cells

The regions are mutually exclusive as they do not have any overlapping areas and collectively exhaustive as every location in the network space is associated with at least one generator. Therefore, an *exact approximation* of the network Voronoi diagram can be obtained by using a *region quad-tree* [18] where the leaf nodes of the quad-tree correspond to a region in a Voronoi cell in NVD. In particular, with VQ-tree the root node represents the rectangular region enclosing the entire span of the road network (and hence NVD) under consideration. We subdivide this rectangular region into four equal quadrants where each quadrant is one of the four child nodes of the root. Subsequently, we recursively subdivide the quadrants until each quadrant contains only one network Voronoi cell information. That is, for each quadrant, we search for two (or more) different color-coded nodes[2]. If we find such a quadrant (meaning that the quadrant includes more than one network Voronoi cell), we subdivide that quadrant into four subquadrants. This subdivision process continues recursively until all nodes in a quadrant have the same color code.

Figure 8 illustrates the quad-blocks generated on the road network in Figure 4. We note that the leaf nodes of VQ-tree does not store any information about the network nodes. As shown in Figure 9, the leaf nodes only store the region information (i.e., coordinates) of the quad-blocks as well as a single value (e.g, a color code or a integer number) which indicates the identity of the network Voronoi cell of which the quad-tree block is a member. We note that a leaf node in the quad-tree corresponds to a particular subdivision of a network Voronoi cell.

As shown in 8, each network Voronoi cell NVC_i consists of disjoint quad-tree blocks. The disjoint decomposition of the network Voronoi diagram with VQ-tree addresses the two drawbacks of VR-tree. Specifically, unlike VR-tree that roughly estimates the network Voronoi cells with polygons in the Euclidean space, VQ-tree enables the exact representation of the network Voronoi cells

[2] During NVD construction parallel Dijkstra algorithm can encode each node with a Voronoi cell identifier, e.g., a color.

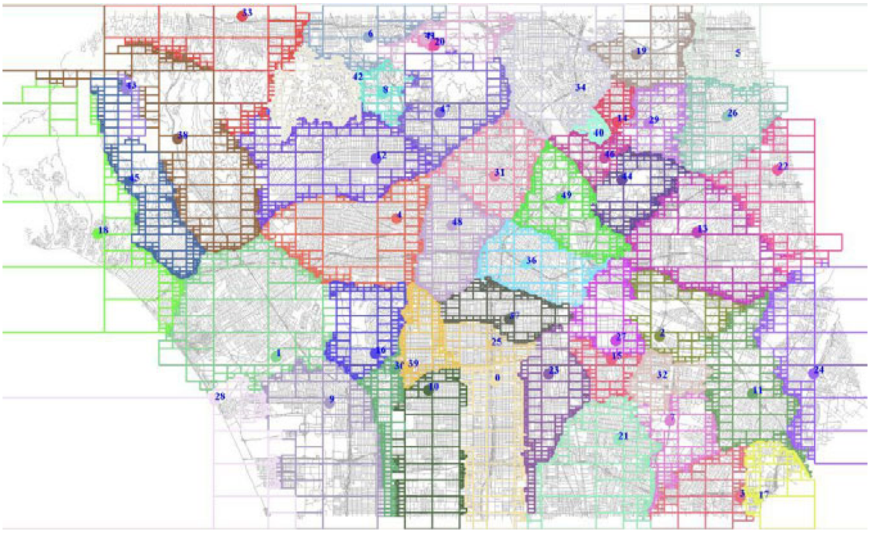


Fig. 8. VQ-tree on Los Angeles road network

using quad-tree blocks and hence always yield correct results. VQ-tree does not suffer from the backtracking problem of VR-tree, and hence fast response time for *contain*(q). This is due to non-overlapping partitioning of the network Voronoi cells: once the quad-tree block containing q is located in the leaf nodes, VQ-tree immediately identifies the nearest Voronoi generator based on the value (e.g, a color code) of that block.

Algorithm 1 presents the outline for VQ-tree. Given a set of N nodes with their color codes and bounding box $[x_1; x_2]$x$[y_1; y_2]$ that contains N as an input, Algorithm 1 creates VQ-tree by recursively splitting the quadrants until all the nodes in a quadrant have the same color code.

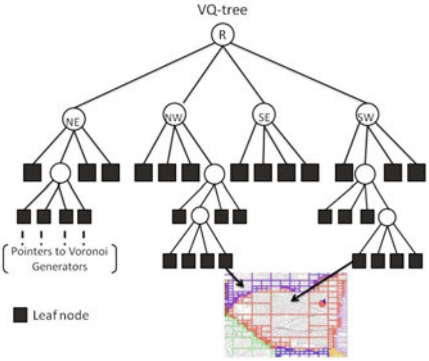


Fig. 9. VQ-tree

Algorithm 1. VQ-Tree Algorithm

```
$VQuadTree(N, x_1, x_2, y_1, y_2)\{$
/* Scan distinct color codes in the region
$cellColor[] \Leftarrow checkRegion(N, x_1, x_2, y_1, y_2)$;
/* If there exist more than one color-code then split
if $cellColor.length > 1$ then
    /*Initialize intermediate node
    $node \Leftarrow QuadTreeNode()$;
    /*Set Quadrants
    $node.SE \Leftarrow VQuadTree(N, x_1, (x_2{+}x_1)/2, y_1, (y_1{+}y_2)/2)$;
    $node.SW \Leftarrow VQuadTree(N, (x_2{+}x1)/2, x_2, y_1, (y_1{+}y_2)/2)$;
    $node.NE \Leftarrow VQuadTree(N, x_1, (x_2{+}x_1)/2, (y_1{+}y_2)/2, y_2)$;
    $node.NW \Leftarrow VQuadTree(N, (x_2{+}x_1)/2, x_2, (y_1{+}y_2)/2, y_2)$;
else
    /*Create leaf node
    $QuadTreeLeafNode(cellColor[0])$;
end if
$\}$
```

5 Experimental Evaluation

5.1 Experimental Setup

We conducted experiments with different spatial networks and various parameters to evaluate the performance of VQ-tree and VR-tree. We measured the ratio of false-negative edges with varying object cardinality (i.e., number of Voronoi generators) and object distribution in the road network. In addition, we compared the precomputation, index rebuilding (for dynamic environments) and response time of VQ-tree and VR-tree with respect to different network sizes and object cardinality. As of our dataset, we used California (CA), Los Angeles (LA) and San Joaquin County (SJ) road network data (obtained from Navteq [10]) with approximately 1,965,300, 304,162 and 24,123 nodes, respectively. Since the experimental results with LA and SJ networks differ insignificantly, we only present the results from the CA and LA datasets. We conducted our experiments on a workstation with 2.7 GHz Pentium Core Duo processor and 12GB RAM memory. For each set of experiments, we only vary one parameter and fix the remaining to the default values in Table 1.

Table 1. Experimental parameters

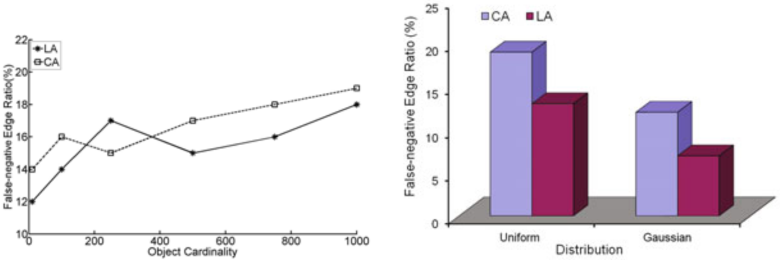
Parameters	Default	Range
Object Cardinality	100	10,50,100,500,1000
Road Network	LA	SJ, LA, CA
Object Distribution	Uniform	Uniform, Gaussian

5.2 Results

5.2.1 Ratio of False-Negative Edges

First, we study the ratio of false-negative edges with respect to object cardinality (i.e., number of Voronoi generators) and object distribution. To identify false-negative edges, we compare the encoded values (i.e., color code) of each node based on VR-tree and VQ-tree. Specifically, we first encode each edge to its corresponding Voronoi generator by using VR-tree polygons and then compare the encoded values to that we obtained from VQ-tree. We repeat each experiment 100 times and report the average number of incorrectly encoded (i.e., false-negative) edges with respect to total number of edges in the network. Figure 10(a) shows the ratio of false-negative edges of both networks where the object cardinality ranging from 10 to 1000. As illustrated, the ratio of incorrectly identified edges is %16 on average in both networks. The maximum recorded false-negative edge ratio for LA and CA road networks is %24 and %29, respectively.

Figure 10(b) illustrates the ratio of false-negative edges with different object distribution for both CA and LA road networks. We observe that the number of false-negative edges is less in Gaussian distribution. This is because as objects are clustered in the spatial network with Gaussian distribution, the corresponding shortest path trees would be less disperse and hence spatially close border points. As mentioned, with VR-tree we encode the edges based on the Euclidean polygon generated by connecting the border points. The more spatially close border points provides the more accurate presentation of the NBCs and hence less false-negative edges.



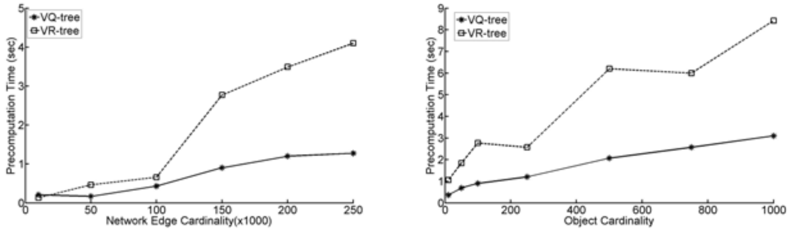
(a) Impact of object cardinality (b) Impact of object distribution

Fig. 10. Impact of object cardinality and distribution

5.2.2 Precomputation Time

With another set of experiments, we compare the precomputation (i.e., index construction) time of VR-tree and VQ-tree with varying network sizes and number of objects. In order to evaluate the impact of network size, we conducted experiments with the sub-networks of CA dataset ranging from 50K to 250K segments. We set the the node size of VR-tree to 4K bytes in all cases. Figure 11(a) shows the precomputation time of VQ-tree and VR-tree in CA road network with varying network size. The results indicate that the precomputation time increases with the network size in both methods where VQ-tree outperforms VR-tree with all numbers of edges. This is because as the network size increases the perimeters of the polygons (and hence the number of connected line segments that form a polygon) grow in VR-tree. Arguably, the overhead of generating MBRs (to be used in VR-tree) around the polygons composed of numerous connected line strings is time-consuming as the coordinates (that form the lines) needs to be scanned to find the ultimate corners of the MBR. On the other hand, VQ-tree is constructed based the underlying space (rather than objects in VR-tree) by recursively dividing the road network to quad-blocks each corresponding to one NVC.

Figure 11(b) illustrates the impact of object cardinality over precomputation time in LA road network (the results are similar in CA network and hence not presented). We observe that as the number of objects in the road network increases, the preprocessing time for both approaches increases. As shown, the precomputation time for VQ-tree outperforms VR-tree. The reason is that the time for hierarchically clustering polygons in VR-tree for a large datasets is relatively expensive. We also observe that the depth of VQ-tree increases with the increasing number of data objects. This is because large number of data objects yields smaller VCs and hence more splits.

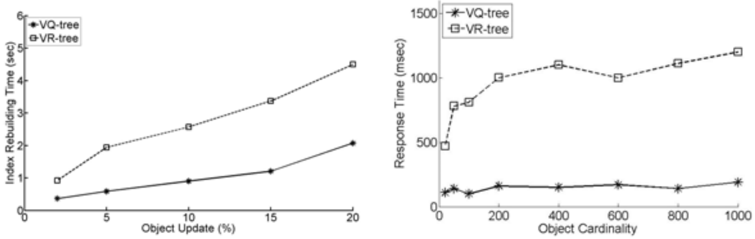


(a) Impact of network size (b) Impact of object cardinality

Fig. 11. Impact of network size

5.2.3 Index Reconstruction

Next, we compare the index reconstruction overhead of VR-tree and VQ-tree with respect to object updates. In this set of experiments, we update the location of the randomly selected data objects and measure the index reconstruction overhead in both VR-tree and VQ-tree. Figure 12(a) shows the index reconstruction time of both index structures with varying object update ratio (i.e., the percentage of data objects whose locations changed). We observe that VQ-tree outperforms VR-tree with respect to index reconstruction. This is because the insert operations in VR-tree are expensive. When new data objects are inserted into VR-tree, besides updating leaf nodes, it is likely that updates are also required to non-leaf nodes (i.e., more than one branch of the tree maybe expanded), which leads to a large overhead during insertion. On the other hand, with VQ-tree we observe that most of the index updates take place in the leaf nodes.



(a) Index reconstruction

(b) Impact of object cardinality

Fig. 12. Response time vs object cardinality and Index reconstruction

5.2.4 Response Time

In this experiment, we compare the performance (i.e., the response time for *contain*(q) query) of VQ-tree and VR-tree with varying object cardinality. We determine the location of the query object q uniformly at random and report average of 100 queries. As we mentioned the original VR-tree proposed in [7] may yield inaccurate results. In order to provide correct results with VR-tree, we modify VR-tree by adding an additional index structure that maintains false-negative edges. Specifically, we construct a R-tree on the false-negative network edges along with their Voronoi generators. With each *contain*(q) query, we check q against this index structure. If we locate q on any of the false-negative edges, the corresponding data object is returned as the first NN. Otherwise, VR-tree continues the search based on the polygons explained in 4.2.1. Figure 12(b) plots the average response time for *contain*(q) query. The results indicate that VQ-tree outperforms VR-tree and scales better with large number of data objects. The response time of VQ-tree is approximately 12 times better than that of VR-tree with more than 200 data objects. This is because of the fact that, with VR-tree, the amount of work often depends on the size of the overlapping areas. In particular, the overlapping areas may belong to more than one NVC and hence during the search the parent node(s) of the overlapping MBRs have to be accessed repeatedly. We also implemented VR-tree using R+ tree (VR+) that minimizes the impact of overlapping areas. We observe that the performance of VQ-tree is still 7 times superior to VR+ tree.

6 Conclusion

In this paper, we study two different spatial index structures, namely the Voronoi R-tree and Voronoi Quad-tree, to index network Voronoi diagrams. These index structures enable efficient access to the network Voronoi cells containing a particular point or edge of the network. We show that previously proposed Voronoi R-tree may yield inaccurate results and fail to scale in large road networks with numerous data objects. We propose a novel approach, termed Voronoi Quad-tree, that enables disjoint decomposition of the network Voronoi diagram where network Voronoi cells are indexed with region quad-tree. The precomputation overhead of the Voronoi Quad-tree is significantly less and the Voronoi Quad-tree outperforms Voronoi R-tree in query response time by a factor of 1:4 to 12 depending on the network size and object cardinality. We intend to pursue this study in two directions. First, we plan to investigate disk organization strategies for Voronoi Quad-tree. Second, we intend to work on incremental index update techniques to avoid node reconstruction overhead due to update in the location of Voronoi generators.

References

1. Cho, H.-J., Chung, C.-W.: An efficient and scalable approach to cnn queries in a road network. In: VLDB (2005)
2. Erwig, M., Hagen, F.: The graph voronoi diagram with applications. Journal of Networks 36 (2000)
3. Finkel, R.A., Bentley, J.L.: Quad trees: A data structure for retrieval on composite keys. Acta Informatica (1974)
4. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: SIGMOD (1984)
5. Hu, H., Lee, D.L., Xu, J.: Fast Nearest Neighbor Search on Road Networks. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Böhm, K., Kemper, A., Grust, T., Böhm, C. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 186–203. Springer, Heidelberg (2006)
6. Jensen, C.S., Kolářvr, J., Pedersen, T.B., Timko, I.: Nearest neighbor queries in road networks. In: GIS (2003)
7. Kolahdouzan, M., Shahabi, C.: Voronoi-based k nearest neighbor search for spatial network databases. In: VLDB (2004)
8. Kolahdouzan, M.R., Shahabi, C.: Continuous k-nearest neighbor queries in spatial network databases. In: STDBM (2004)
9. Mokbel, M.F., Xiong, X., Aref, W.G.: Sina: scalable incremental processing of continuous queries in spatio-temporal databases. In: SIGMOD (2004)
10. NAVTEQ, www.navteq.com (accessed in May 2011)
11. Nutanong, S., Tanin, E., Ali, M.E., Kulik, L.: Local network voronoi diagrams. In: SIGSPATIAL (2010)
12. Okabe, A., Boots, B., Sugihara, K., Chiu, S.N.: Spatial tessellations — concepts and applications of voronoi diagrams (2000)
13. Okabe, A., Satoh, T., Furuta, T., Suzuki, A., Okano, K.: Generalized network voronoi diagrams: Concepts, computational methods, and applications. Int. J. Geogr. Inf. Sci. (2008)
14. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: VLDB (2003)
15. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: SIGMOD (1995)
16. Safar, M.: Group-nearest neighbors queries in spatial network databases. Journal of Geographical Systems (2008)
17. Safar, M., Ibrahimi, D., Taniar, D.: Voronoi-based reverse nearest neighbor query processing on spatial networks. Multimedia Systems (2009)
18. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan-Kaufmann, San Francisco (2006)

19. Samet, H., Sankaranarayanan, J., Alborzi, H.: Scalable network distance browsing in spatial databases. In: SIGMOD (2008)
20. Sellis, T.K., Roussopoulos, N., Faloutsos, C.: R+-tree: A dynamic index for multidimensional objects. In: VLDB 1987 (1987)
21. Song, Z., Roussopoulos, N.: K-Nearest Neighbor Search for Moving Query Point. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) SSTD 2001. LNCS, vol. 2121, pp. 79–96. Springer, Heidelberg (2001)
22. Taniar, D., Safar, M., Tran, Q.T., Rahayu, J.W., Park, J.H.: Spatial network rnn queries in gis. Comput. J. (2011)
23. Tao, Y., Papadias, D., Lian, X.: Reverse knn search in arbitrary dimensionality. In: VLDB (2004)
24. Tao, Y., Papadias, D., Shen, Q.: Continuous nearest neighbor search. In: VLDB (2002)
25. Huang, X., Jensen, C.S., Šaltenis, S.: The Island Approach to Nearest Neighbor Querying in Spatial Networks. In: Medeiros, C.B., Egenhofer, M., Bertino, E. (eds.) SSTD 2005. LNCS, vol. 3633, pp. 73–90. Springer, Heidelberg (2005)
26. Xuan, K., Zhao, G., Taniar, D., Rahayu, J.W., Safar, M., Srinivasan, B.: Voronoi-based range and continuous range query processing in mobile databases. J. Comput. Syst. Sci. (2011)
27. Xuan, K., Zhao, G., Taniar, D., Srinivasan, B., Safar, M., Gavrilova, M.: Network voronoi diagram based range search. In: Advanced Information Networking and Applications
28. Yiu, M.L., Mamoulis, N., Papadias, D.: Aggregate nearest neighbor queries in road networks. In: ICDE (2005)
29. Yiu, M.L., Papadias, D., Mamoulis, N., Tao, Y.: Reverse nearest neighbors in large graphs. In: ICDE (2005)
30. Zhang, J., Zhu, M., Papadias, D., Tao, Y., Lee, D.L.: Location-based spatial queries. In: SIGMOD (2003)
31. Zhao, G., Xuan, K., Taniar, D., Safar, M., Gavrilova, M., Srinivasan, B.: Multiple Object Types KNN Search Using Network Voronoi Diagram. In: Gervasi, O., Taniar, D., Murgante, B., Laganà, A., Mun, Y., Gavrilova, M.L. (eds.) ICCSA 2009, Part II. LNCS, vol. 5593, pp. 819–834. Springer, Heidelberg (2009)
32. Zheng, B., Lee, D.L.: Semantic Caching in Location-Dependent Query Processing. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) SSTD 2001. LNCS, vol. 2121, pp. 97–113. Springer, Heidelberg (2001)

On Efficient Reverse *k*-Skyband Query Processing

Qing Liu[1], Yunjun Gao[1], Gang Chen[1], Qing Li[2], and Tao Jiang[3]

[1] College of Computer Science, Zhejiang University
{liuqing1988,gaoyj,cg}@zju.edu.cn
[2] Department of Computer Science, City University of Hong Kong
itqli@cityu.edu.hk
[3] College of Mathematics Physics and Information Engineering, Jiaxing University
jiangtao_guido@yahoo.com.cn

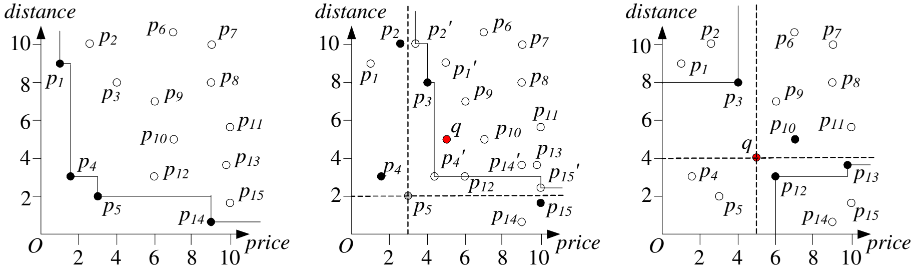
Abstract. This paper, for the first time, addresses the problem of efficient *reverse k-skyband* (R*k*SB) query processing. Given a set P of multi-dimensional points and a query point q, an R*k*SB query returns all the points in P whose *dynamic k-skyband* contains q. We formalize the R*k*SB query, and then propose three algorithms for computing the R*k*SB of an *arbitrary* query point efficiently. Our methods utilize a conventional data-partitioning index (e.g., R-tree) on the dataset, as well as employ *pre-computation* and *pruning* techniques to improve the query performance. Extensive experiments using both real and synthetic datasets demonstrate the effectiveness of our proposed pruning heuristics and the performance of our proposed algorithms.

1 Introduction

Given a set P of multi-dimensional points, a *traditional/static skyline query* [2, 4, 7, 9, 12, 16, 20] returns all the points in P that are *not dominated* by any other point. A point p *dominates* another point p' if p is not worse than p' in all dimensions and strictly better than p' in at least one dimension. Figure 1(a) shows a classical example of static skyline over a hotel dataset $S_h = \{p_1, p_2, \ldots, p_{15}\}$ in a 2-dimensional (2D) space, where the x-axis represents the room *price* of each hotel and the y-axis captures its *distance* to the beach. Since hotels p_1, p_4, p_5, and p_{14} are not dominated by others, they constitute the static skyline of the dataset S_h.

In contrast, the *dynamic skyline query* [5, 12, 15] is a natural extension of the static skyline query, where the attributes of every point are dynamically calculated w.r.t. ad hoc functions or query points. Specifically, each d-dimensional point p is mapped to a new m-dimensional point $p' = \langle f_1(p), f_2(p), \ldots, f_m(p)\rangle$, in which f_i ($i \in [1, m]$) is a *dimension function*. Like [5, 10], in this paper, we assume that $m = d$, and $f_i(p) = |p[i] - q[i]|$ for a specified query point q. In Figure 1(b), for instance, p_3 remains the same (as it is still in the first quadrant) and p_4 is mapped to p_4', assuming that p_5 is a query point. The dynamic skyline query is to retrieve all the points in P that are not *dynamically dominated*, w.r.t. a given query point q, by any other point. A point p dynamically dominates another point p' w.r.t. q, if for all dimensions p is closer to q than p' and it is strictly closer to q than p' in at least one dimension. Figure 1(b)

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 544–559, 2012.
© Springer-Verlag Berlin Heidelberg 2012



(a) Static skyline (b) Dynamic skyline (c) Reverse skyline

Fig. 1. Illustration of static, dynamic, and reverse skylines

illustrates the dynamic skyline query w.r.t. p_5, as well as points p_2, p_3, p_4, and p_{15} are the *dynamic skyline points* of p_5.

As shown in Figure 1(b), a point q is not in the dynamic skyline of p_5. In general, if q belongs to the dynamic skyline of a certain point p, p is said to be in the *reverse skyline* [5, 10, 14, 17, 18, 22] of p. A reverse skyline query finds all the points in P that have a given query point q as a member of their dynamic skylines. For example, Figure 1(c) depicts the reverse skyline query w.r.t. q, and the result includes p_3, p_{10}, p_{12}, and p_{13}. The reverse skyline operator is useful for many applications such as business location planning [5, 18] and environmental monitoring [10, 17]. However, as pointed out in [5], it may obtain too *few* reverse skyline points to choose from. Although the user might change the location of the query point to increase the number of reverse skyline points, finding a good query point location is far from easy, since many factors can ultimately affect the query performance.

In this paper, we introduce a new operator, namely *reverse k-skyband* (R*k*SB) *query*. Given a multi-dimensional data set P and a query point q, an R*k*SB query returns all the points in P whose *dynamic k-skyband* (defined in Definition 1) contains q. It is helpful for many applications. As an example, suppose the decision-maker of a computer manufacturer wants to know how many customers may be interested in a new/forthcoming computer. In this case, he/she can take the new computer as a query point q and the potential customer preferences (e.g., CPU, main-memory, etc.) as a dataset P, and then perform several R*k*SB queries for marketing analysis. If the k is big enough while the query result is still small, the decision-maker has to change the computer setting.

Actually, the R*k*SB query is the generalization of reverse skyline query. Thus, the existing methods [5] specifically designed for reverse skyline queries are not (directly) applicable to tackle the R*k*SB query efficiently. In this paper, we propose three efficient algorithms, i.e., *Branch-bound-based RkSB algorithm* (BR*k*SB), *Pre-computation-based RkSB algorithm* (PR*k*SB), and *Optimized PRkSB* (OPR*k*SB), to compute the R*k*SB of an *arbitrary* query point. Our approaches utilize a conventional data-partitioning index (e.g., R-tree [1]) on the dataset, and employ *offline pre-computation* and *pruning* techniques to improve the query performance. In brief, the key contributions of this paper are summarized as follows:

- We formalize the R*k*SB query, an interesting variant of reverse skyline query. To the best of our knowledge, this work is the first attempt on this problem.
- We develop three algorithms, viz. BR*k*SB, PR*k*SB and OPR*k*SB, for efficiently answering reverse *k*-skyband queries, and analyze their correctness. Specifically, BR*k*SB is an improved customization of the original BBS algorithm [12], PR*k*SB utilizes accurate pre-computed dynamic skylines to discard unnecessary candidates, and OPR*k*SB enables effective pruning heuristics to prune away unqualified candidates.
- We conduct extensive experiments using both real and synthetic datasets to demonstrate the effectiveness of our proposed heuristics and the performance of our proposed algorithms.

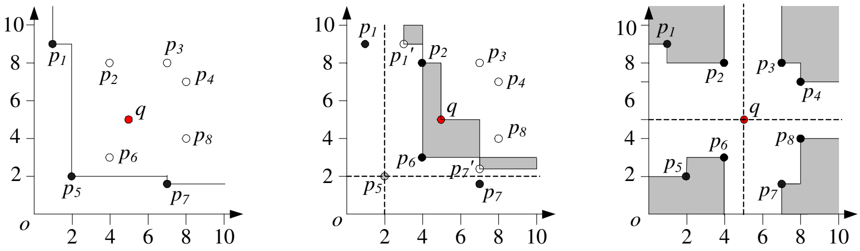
The rest of the paper is organized as follows. Section 2 briefly surveys the related work. Section 3 presents our problem statement. Section 4 elaborates three R*k*SB query processing algorithms and discusses their correctness. Extensive experimental evaluation and our findings are reported in Section 5. Finally, Section 6 concludes the paper with some directions for future work.

2 Related Work

The skyline query is a popular paradigm for extracting interesting objects from multi-dimensional databases. Borzsony et al. [2] first introduce the skyline operator in the database community, and develop two skyline computation methods, i.e., *Block Nested Loop* (BNL) and *Divide-and-Conquer* (D&C). Chomicki et al. [4] present a *Sort-First-Skyline* (SFS) algorithm as an improved version of BNL. Godfrey et al. [7] propose an optimized version of SFS, termed *Linear-Elimination-Sort for Skyline* (LESS), which has attractive worst-case asymptotical performance. Zhang et al. [20] present an *object-based space partitioning* (OSP) scheme for scalable skyline computation. These algorithms do not assume any index on the dataset.

On the other hand, other approaches exploit indexes to accelerate skyline queries. Tan et al. [16] first propose the *progressive* technique that can return skyline points instantly, and develop two approaches for skyline queries, namely *Bitmap* and *Index*, respectively. Another two progressive skyline query algorithms, i.e., *Nearest Neighbor* (NN) and *Branch-and-Bound Skyline* (BBS), are proposed by Kossmann et al. [9] and Papadias et al. [12], respectively. As demonstrated in [12], BBS is I/O-optimal algorithm. Recently, numerous useful variations of skyline queries have been studied as well. Examples include *dynamic skyline query* [12, 15], *constrained skyline computation* [3, 12], *reverse skyline query* [5, 10], *probabilistic skyline retrieval* [13, 21], *parallel skyline query* [6, 8, 19], *stochastic skyline operator* [11], and so on.

The concept of reverse skyline is originally introduced in [5]. In order to compute the reverse skyline of an arbitrary query point, Dellis and Seeger [5] propose two algorithms: *Branch and Bound Reverse Skyline algorithm* (BBRS) and *Reverse Skyline using Skyline Approximations algorithm* (RSSA). In particular, BBRS is an improved customization of the BBS algorithm [12], while RSSA is based on the well-known *filter-refinement* paradigm, and employs pre-computed approximations of the skylines. Lian and Chen [10] study monochromatic and bichromatic reverse skyline search on *uncertain* data, where each object is modeled as a probability distribution function. Wu et al. [18] explore bichromatic reverse skyline retrieval for traditional dataset, in which each object is a *precise* point. More recently, techniques for reverse skyline computation over wireless sensor networks [17], data streams [22], and arbitrary non-metric similarity measures [14] have also been proposed in the literature. In this paper, we focus on the problem of *reverse k-skyband* retrieval.



(a) Skyline (b) Dynamic 1-skyband of p_5 (c) Reverse 1-skyband of q

Fig. 2. Illustration of skyline, dynamic 1-skyband, and reverse 1-skyband

3 Problem Formulation

In this section, we present the definition of dynamic *k*-skyband, and then formally define the reverse *k*-skyband query.

Let P be a d-dimensional dataset. For any point $p \in P$, we use $p[i]$ to denote the i-th dimensional value of p. A point $p \in P$ is said to *dominate* another point $p' \in P$, denoted as $p \prec p'$, if (i) for every $i \in \{1, 2, \ldots, d\}$, $p[i] \leq p'[i]$; and (ii) for at least one $j \in \{1, 2, \ldots, d\}$, $p[j] < p'[j]$. For instance, in figure 2(a), point p_5 dominates point q.

Definition 1 (Dynamic *k*-Skyband). *Given a d-dimensional data set P, a query point q, and a parameter k, if a point p* ∈ *P belongs to the* dynamic *k*-skyband *of q, there exist* at most *k points in P, denote by O, such that for each point o* ∈ *O, it satisfies*: (1) *for any* $i \in \{1, 2, \ldots, d\}$, $|q[i] - o[i]| \leq |q[i] - p[i]|$; *and* (2) *at least one* $j \in \{1, 2, \ldots, d\}$, $|q[j] - o[j]| < |q[j] - p[j]|$.

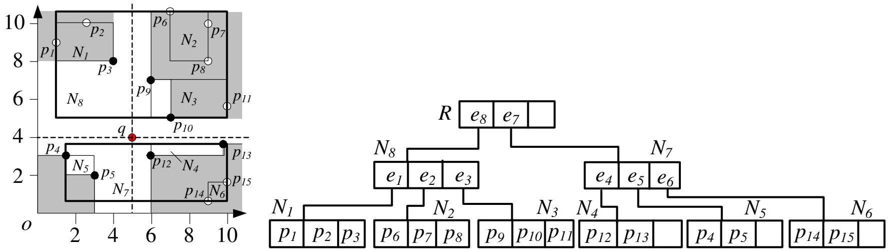
A dynamic *k*-skyband query retrieves all the points that are dynamically dominated by *at most k* points. Figure 2(b) illustrates the dynamic 1-skyband of point p_5. As shown in Figure 2(b), although the point q is not the dynamic skyline point of p_5, it is in p_5's dynamic 1-skyband since the number of points which dynamically dominate q is *no more than* 1. Note that dynamic *k*-skyband is a generalization of dynamic skyline [12].

Based on the dynamic *k*-skyband, we now formalize the reverse *k*-skyband query.

Definition 2 (Reverse *k*-Skyband Query). *Given a d-dimensional data set P, a query point q, and a parameter k, a* reverse *k*-skyband (R*k*SB) query *returns all the points in P whose* dynamic *k*-skyband *contains q. Formally, a point p* ∈ *P is in the reverse* *k-skyband of q, if there exist* at most *k points in P, denoted as O, such that for every point o* ∈ *O, it holds*: (1) *for any i* ∈ {1, 2, …, *d*}, $|p[i] - o[i]| \leq |p[i] - q[i]|$; *and* (2) *at least one j* ∈ {1, 2, …, *d*}, $|p[j] - o[j]| < |p[j] - q[j]|$.

Take Figure 2(b) as an example. Since a point q is included in the dynamic 1-skyband of point p_5, the point p_5 belongs to the reverse 1-skyband of q according to Definition 2. Similarly, we can obtain the complete reverse 1-skyband of q, which contains $\{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$, as depicted in Figure 2(c). It is worth mentioning that, conceptually, k represents the *thickness* of the reverse skyline. Therefore, the case $k = 0$ corresponds to a conventional reverse skyline query.

A naive solution to tackle R*k*SB retrieval is to perform the dynamic *k*-skyband query for each point in a specified data set P, and then return those points $p \in P$ with $q \in DSB(p)$, where $DSB(p)$ represents the set of p's dynamic *k*-skyband points. Nevertheless, this method is *very inefficient* as it needs to traverse the data set *P multiple times* (i.e., $|P|$ times), resulting in *high* I/O and CPU costs, especially for *larger P*. Motivated by this, in the next section, we will propose three efficient algorithms for processing the R*k*SB query.



(a) Data point placement (b) The corresponding R-tree

Fig. 3. A running example

4 R*k*SB Query Processing

In this section, we first propose three algorithms for efficiently computing the R*k*SB of an arbitrary query point, assuming that the data set P is indexed by an R-tree, and then provide an analysis of these algorithms. In order to facilitate the understanding of different R*k*SB query processing algorithms, a running example, as shown in Figure 3, is employed. Specifically, we use the 2D data point set $P = \{p_1, p_2, \ldots, p_{15}\}$ of Figure 3(a), organized in the R-tree of Figure 3(b) with node capacity = 3.

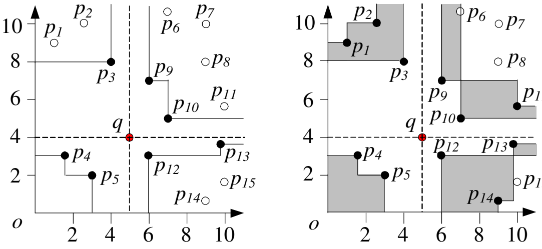
4.1 Branch-Bound-Based Reverse *k*-Skyband Algorithm

In this subsection, we describe the *Branch-bound-based Reverse k-Skyband algorithm* (BR*k*SB), which is an improved customization of the original BBS algorithm. Next, we define the *global k-skyband*, and then present two lemmas to identify/discard candidate R*k*SB points.

Definition 3 (Global *k*-Skyband). *Given a d-dimensional data set P, a query point q, and a parameter k, if a point $p \in P$ is in the* global k-skyband *of q, there exist* at most *k points in P, denote as O, such that for each point $o \in O$, it satisfies*: (1) *for any* $i \in \{1, 2, \ldots, d\}$, $(p[i] - q[i])\ (o[i] - q[i]) > 0$; (2) *for any* $i \in \{1, 2, \ldots, d\}$, $|q[i] - o[i]| \leq |q[i] - p[i]|$; *and* (3) *for at least one* $j \in \{1, 2, \ldots, d\}$, $|q[j] - o[j]| < |q[j] - p[j]|$.

A global k-skyband query finds all the points that are globally dominated by at most k points. Note that global k-skyband is an extension of global skyline [5]. Figure 4 shows an example of the global skyline and global 1-skyband for point q. Since point p_8 is globally dominated by points p_9 and p_{10}, it is not a global 1-skyband point of q.

Lemma 1. *Let q be a query point, GSB(q) the global k-skyband point set of q, and RSB(q) the set of reverse k-skyband points w.r.t. q. If a point $p \notin GSB(q)$, $p \notin RSB(q)$.*



(a) Global skyline (b) Global 1-skyband

Fig. 4. Illustration of global skyline and global k-skyband

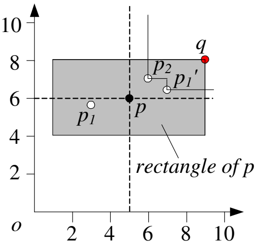


Fig. 5. Example of Lemma 2

Proof. If a point $p \notin GSB(q)$, we can retrieve $(k + 1)$ points (stored in a set S) that globally dominate p, that is, for every point $s \in S$, it holds: for all $i \in \{1, 2, \ldots, d\}$, $|s[i] - q[i]| \leq |p[i] - q[i]|$; and there exists a $j \in \{1, 2, \ldots, d\}$, $|s[j] - q[j]| < |p[j] - q[j]|$, which contradicts with Definition 2 (i.e., $p \notin RSB(q)$). The proof completes. □

We can utilize Lemma 1 to efficiently retrieve a *superset* of the actual result, i.e., *no false misses*. The Lemma 2 below helps us to prune away unqualified candidate reverse k-skyband points that cannot be the final answer points. □

Lemma 2. *Given a d-dimensional data set P, a query point q, and a parameter k. Suppose a rectangle Rect is centered at a point $p \in P$, and its extent is defined by the coordinate-wise distances to q. If there exist* more than *k points within Rect, p is not a reverse k-skyband point of q.*

Proof. If there are more than k points inside the rectangle *Rect*, we can find $(k + 1)$ points (preserved in a set S) that, for each point $s \in S$, satisfy: for all $i \in \{1, 2, \ldots, d\}$, $|p[i] - s[i]| \leq |p[i] - q[i]|$; and there exist $j \in \{1, 2, \ldots, d\}$, $|p[j] - s[j]| < |p[j] - q[j]|$. According to Definition 1, the point q does not belong to the dynamic k-skyband of the point p. Thus, p cannot be a reverse k-skyband point of q. □

Consider, for instance, Figure 5, in which the shaded area is the rectangle of a point *p*. As shown in Figure 5, a point *q* is dynamically dominated by points p_1 and p_2, which are located inside *p*'s rectangle. Consequently, the point *q* is not in the dynamic 1-skyband of the point *p*, and *p* does not belong to the reverse 1-skyband of *q*.

Our first method, namely BR*k*SB, uses the above lemmas to process R*k*SB query. The basic idea is as follows. First, BR*k*SB computes the set *GSB*(*q*) of global *k*-skyband points that is guaranteed to include all the actual reverse *k*-skyband points, using Lemma 1. Then, the algorithm executes a window query for each candidate global *k*-skyband point, and the candidate is a true reverse *k*-skyband point (based on Lemma 2) if the window query returns no more than *k* points. The pseudo-code of BR*k*SB is presented in Algorithm 1. Next, we discuss BR*k*SB using the running example (depicted in Figure 3) for computing the reverse 1-skyband. The query result contains points p_3, p_4, p_5, p_9, p_{10}, p_{12}, and p_{13}.

Initially, BR*k*SB visits the root of the R-tree *R* and inserts all its entries (e_7, e_8) into a heap H_g (line2). Then, the entry (e_7) with the minimum distance to *q* is expanded. As e_7 is not globally dominated by any point (line 5) and it is an intermediate node, BR*k*SB removes the entry (e_7) from the heap and adds its children (e_4, e_5, e_6) to H_g (line 8-11). Similarly, the next two expanded entries are e_8 and e_4 respectively, in which the first global 1-skyband point p_{12} is found. Since p_{12} is a data point, BR*k*SB adds p_{12} to the set *GSB* for pruning later, and then runs a window query on it (lines 13-14). As shown in Figure 6, there is *no* point in the window/rectangle centered at p_{12}. Thus, p_{12} belongs to the reverse 1-skyband of *q* and is inserted into the result set *RSB*. The algorithm proceeds in the same manner until the heap becomes empty. Table 1 shows the contents of the heap during the processing of the query.

Algorithm 1. Branch-bound-based Reverse *k*-Skyband Algorithm (BR*k*SB)

```
Input: an R-tree R on a set of data points, a query point q, a parameter k
Output: the result set RSB that contains all the points belonging to the RkSB of q
/* GSB: the set of global k-skyband points; RSB: the set of reverse k-skyband points; $H_g$, $H_w$:
   min-heaps, sorted in ascending order of their distances (i.e., $L_1$-norm) from q. */
1:  initialize sets $GSB = RSB = \varnothing$ and min-heaps $H_g = H_w = \varnothing$
2:  insert all entries of the root R into $H_g$
3:  while $H_g \neq \varnothing$ do
4:      de-heap the top entry e of $H_g$
5:      if e is globally dominated by $(k + 1)$ points in GSB then
6:          discard e    // by Lemma 1
7:      else    // e is globally dominated by less than $(k + 1)$ points in GSB
8:          if e is an intermediate node then
9:              for each child entry $e_i \in e$ do
10:                 if $e_i$ is globally dominated by at most k point in GSB then
11:                     insert $e_i$ into $H_g$
12:         else    // e is a data point
13:             add e to GSB    // for pruning later
14:             perform the window query (using $H_w$) based on e and q
15:             if the window query finds less than $(k + 1)$ points then
16:                 add e to RSB    // e is a RkSB point of q by Lemma 2
17:             else    // the window query contains more than k points
18:                 discard e
19: return RSB
```

4.2 Pre-computation-Based Reverse *k*-Skyband Algorithm

Since the global *k*-skyband may contain many unnecessary points, we present, in this section, an enhanced algorithm, called *Pre-computation-based Reverse k-Skyband algorithm* (PR*k*SB). Specifically, PR*k*SB employs the traditional (i.e., 0-th) dynamic skyline and the *k*-th dynamic skyline, which are *offline* pre-computed and stored on disk, to identify points being in the reverse *k*-skyband and prune unqualified points not belonging to the result. Below, we define the *k-th dynamic skyline*, and then provide Lemma 3 and Heuristic 1 to support our proposed PR*k*SB algorithm.

Definition 4 (*k*-th Dynamic Skyline). *Given a d-dimensional data set P, a query point q, and a parameter k, if a point $p \in P$ is in the k-*th dynamic skyline *of q, there exist k points in P, denoted by O, such that for every point $o \in O$, it holds:* (1) *for any $i \in \{1, 2, \ldots, d\}$, $|q[i] - o[i]| \le |q[i] - p[i]|$; and* (2) *at least one $j \in \{1, 2, \ldots, d\}$, $|q[j] - o[j]| < |q[j] - p[j]|$.*

A *k*-th dynamic skyline query retrieves the set of the points in *P* that are dynamically dominated by *k* points. Figure 7(a) illustrates the example of the *k*-th dynamic skyline, where the 0-th dynamic skyline contains points p_1, p_2, p_5, and p_9; the 1-th dynamic skyline includes points p_3 and p_6; and the 2-th dynamic skyline consists of points p_4 and p_{10}. Note that all of them form the dynamic 2-skyband of *p*.

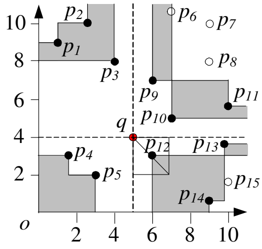
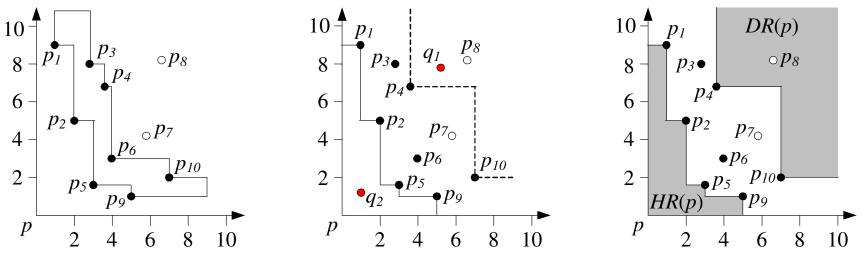


Fig. 6. Window query of p_{12}

Table 1. Heap contents

action	**heap contents**	***rsb***
access root	$\langle e_7, e_8\rangle$	$\varnothing$
expand e_7	$\langle e_8, e_4, e_5, e_6\rangle$	$\varnothing$
expand e_8	$\langle e_4, e_3, e_5, e_1, e_2, e_6\rangle$	$\varnothing$
expand e_4	$\langle \boldsymbol{p_{12}}, e_3, e_5, p_{13}, e_1, e_2, e_6\rangle$	$\{p_{12}\}$
expand e_3	$\langle \boldsymbol{p_{10}}, e_5, p_9, p_{13}, e_1, e_2, p_{11}, e_6\rangle$	$\{p_{12}, p_{10}\}$
expand e_5	$\langle \boldsymbol{p_5}, \boldsymbol{p_9}, \boldsymbol{p_4}, \boldsymbol{p_{13}}, e_1, e_2, p_{11}, e_6\rangle$	$\{p_{12}, p_{10}, p_5, p_9, p_4, p_{13}\}$
expand e_1	$\langle \boldsymbol{p_3}$, ~~e_2, p_{11}~~, $e_6, p_2, p_1\rangle$	$\{p_{12}, p_{10}, p_5, p_9, p_4, p_{13}, p_3\}$
expand e_6	$\langle$~~p_{14}, p_2, p_1~~$\rangle$	$\{p_{12}, p_{10}, p_5, p_9, p_4, p_{13}, p_3\}$



(a) *k*-th Dynamic skyline (b) Lemma 3 (c) *DR*(*p*) and *HR*(*p*)

Fig. 7. Illustration of *k*-th dynamic skyline, lemma 3, and *DR*(*p*) and *HR*(*p*)

Lemma 3. *For a given point p and a query point q, let DSL(p) be the set of 0-th dynamic skyline points of p and kDSL(p) be the set of k-th dynamic skyline points of p. If a point s ∈ DSL(p) is dynamically dominated by q, p belongs to the reverse k-skyband of q. If a point s' ∈ kDSL(p) dynamically dominates q, p is not in the reverse k-skyband of q.*

Proof. If a point $s \in DSL(p)$ is dynamically dominated by q, it means that q is not dynamically dominated by any point. Hence, q is in the dynamic k-skyband of p, and p belongs to the reverse k-skyband of q. If $s' \in kDSL(p)$, there must have k points that dynamically dominate s'. As q is dynamically dominated by s', it is also dynamically dominated by the k points which dynamically dominate s'. Consequently, q is not in the dynamic k-skyband of p, and p does not belong to the reverse k-skyband of q. □

Consider, for example, Figure 7(b), where points q_1 and q_2 are query points. Since point p_4 in the 2-th dynamic skyline of p (i.e., $p_4 \in 2DSL(p)$) dynamically dominates q_1 and q_2 is not dynamically dominated by any point, p belongs to the reverse 2-skyband of q_2 but not the reverse 2-skyband of q_1, according to Lemma 3.

We can efficiently prune some points, using the above Lemma 3. Specifically, when we get a candidate, we can check the candidate whether it is in the two regions,

Algorithm 2. Pre-computation-based Reverse *k*-Skyband Algorithm (PR*k*SB)

```
Input: an R-tree $R$ on a set of data points, a query point $q$, a parameter $k$, the dynamic skyline
       of the dataset, the $k$-th dynamic skyline of the dataset
Output: the result set $RSB$ that contains all the points belonging to the R$k$SB of $q$
/* $DR(p)$: the discard region of $p$ containing the points that are dynamically dominated by at
   least $(k + 1)$ points; $HR(p)$: the hit region of $p$ including the points that are not dynamically
   dominated by any point. */
initialize sets $GSB = RSB = \varnothing$ and min-heaps $H_g = H_w = \varnothing$
insert all entries of the root $R$ into $H_g$
while $H_g \neq \varnothing$ do
    de-heap the top entry $e$ of $H_g$
    if $e$ is globally dominated by $(k + 1)$ points in $GSB$ then
        discard $e$    // by Lemma 1
    else    // $e$ is globally dominated by less than $(k + 1)$ points in $GSB$
        if $e$ is an intermediate node then
            for each child $e_i \in e$ do
                if $e_i$ is globally dominated by at most $k$ point in $GSB$ then
                    insert $e_i$ into $H_g$
        else    // $e$ is a data point
            add $e$ to $GSB$    // for pruning later
            if q is in the HR(e) then    // Heuristic 1
                add $e$ to $RSB$    // $e$ is a R$k$SB point of $q$
            else if q is in the DR(e) then
                discard $e$
            else
                perform the window query (using $H_w$) based on $e$ and $q$
                if the window query finds less than $(k + 1)$ points then
                    add $e$ to $RSB$    // $e$ is a R$k$SB point of $q$ by Lemma 2
                else
                    discard $e$
return $RSB$
```

defined by the 0-th dynamic skyline and the *k*-th dynamic skyline. We assume that the *Discard Region of p* (*DR*(*p*)) contains the points dynamically dominated by at least (k + 1) points, and the *Hit Region of p* (*HR*(*p*)) includes the points not dynamically dominated by any point. Figure 7(c) illustrates an example of *DR*(*p*) and *HR*(*p*).

Heuristic 1. *Given a query point q, a point p, DR*(*p*)*, and HR*(*p*)*. If q falls into DR*(*p*)*, p is not in the reverse k-skyband of q* (*and thus discard it*)*. If q locates inside HR*(*p*)*, p belongs to the reverse k-skyband of q. If q is not within DR*(*p*) *and HR*(*p*)*, p needs to be further validation.*

Our second approach, i.e., PR*k*SB, utilizes the above Heuristic 1 to tackle R*k*SB query. The main idea is as follows. Before the algorithm starts, for every data point the 0-th dynamic skyline and the *k*-th dynamic skyline are *pre-computed* and stored on disk. When a query *q* is issued, PR*k*SB computes its global *k*-skyband (i.e., *GSB*(*q*)) that is an upper bound of the final query result, and then, for each point $p \in$ *GSB*(*q*), checks whether *q* is in *DR*(*p*) or in *HR*(*p*) or not. If *q* is within *DR*(*p*), *p* can be pruned shortly. If *q* is inside *HR*(*p*), *p* is an actual answer point. Otherwise, the algorithm needs to perform a window query on the current candidate point *p* for further refinement. The pseudo-code of PR*k*SB is shown in Algorithm 2.

4.3 Optimized PR*k*SB Algorithm

In this subsection, we propose an improved PR*k*SB algorithm, namely *optimized PRkSB algorithm* (OPR*k*SB), for answering R*k*SB queries. Unlike PR*k*SB, OPR*k*SB employs not only *offline pre-computed dynamic skyline* but also *online computed global k-skyband* to identify the points being in the reverse *k*-skyband as well as filter out those points not being in the reverse *k*-skyband. Like PR*k*SB, a window query is issued for each remaining candidate point, but the number of window queries can be reduced significantly due to effective pruning heuristics, leading to substantial cost savings, as demonstrated in our experimental evaluation. In the following, we present two observations, and then offer Lemma 4 and Heuristic 2 to support our proposed OPR*k*SB algorithm.

From Figure 6, we observe that the points that are not in the same quadrant as the current candidate point evaluated must not in the window. Hence, these points do not need to be considered during the window query. Also, we observe that the global *k*-skyband points are closer to a specified query point, compared against the other points. Thus, they have a high probability of falling into the window.

Lemma 4. *Given a query point q, a data point p, and assume that a rectangle Rect is centered at p and its extent is defined by the coordinate-wise distances to q. If Rect contains* more than *k* global *k*-skyband points, *p is not in the reverse k-skyband of q.*

Proof. Similar to the proof of Lemma 2, and hence omitted. □

In contrary, if there is less than *k* global *k*-skyband points located inside *Rect*, we are not sure whether the current candidate evaluated is the actual answer point or not, since it may contain the points not belonging to the global k-skyband of q if the extent of *Rect* is large enough. Therefore, we still need to run a window query on it.

Heuristic 2. *Given a query point q, a data point p, and assume that a rectangle Rect is centered at p and its extent is defined by the coordinate-wise distances to q. If there exist* more than k global k-skyband points *in Rect, p does not belong to the reverse k-skyband of q and can be discarded; otherwise, it needs to be further evaluation.*

The last method, namely OPRkSB, optimizes PRkSB, using Heuristic 2 above. The basic idea is as follows. OPRkSB computes the global k-skyband of a given query point q, and then uses Heuristic 1 to prune away unqualified global k-skyband points that cannot be the actual answer points. Thereafter, for every remaining candidate point, the algorithm does not directly execute a window query on it, but utilizes the *whole* global k-skyband of q (instead of the remaining global k-skyband) to further filter out it. Algorithm 3 depicts the pseudo-code of OPRkSB.

4.4 Discussion

In this subsection, we analyze the correctness of our proposed algorithms, i.e., BRkSB, PRkSB, and OPRkSB.

Lemma 5. *The three algorithms (viz., BRkSB, PRkSB, and OPRkSB) visit (data point and intermediate) entries of an R-tree in ascending order of their distances to the specified query point q.*

Proof. The proof is straightforward since the algorithm always visits entries according to their *mindist* (i.e., L_1-norm) order preserved by the heap. □

Lemma 6. *Any data point inserted into the result set RSB during the execution of the algorithm is guaranteed to be an actual reverse k-skyband point.*

Algorithm 3. Optimized PRkSB Algorithm (OPRkSB)

```
Input: an R-tree $R$ on a set of data points, a query point $q$, a parameter $k$, the dynamic skyline
        of the dataset, the $k$-th dynamic skyline of the dataset
Output: the result set $RSB$ that contains all the points belonging to the R$k$SB of $q$
/* $PGSB$: the set of global $k$-skyband points after pruned by Heuristic 1. */
initialize sets $GSB = PGSB = RSB = \varnothing$ and min-heaps $H_g = H_w = \varnothing$
insert all entries of the root $R$ into $H_g$
while $H_g \neq \varnothing$ do
    de-heap the top entry $e$ of $H_g$
    if $e$ is globally dominated by $(k + 1)$ points in $GSB$ then
        discard $e$    // by Lemma 1
    else    // $e$ is globally dominated by less than $(k + 1)$ points in $GSB$
        if $e$ is an intermediate node then
            for each child $e_i \in e$ do
                if $e_i$ is globally dominated by at most $k$ point in $GSB$ then
                    insert $e_i$ into $H_g$
        else    // $e$ is a data point
            add $e$ to $GSB$    // for pruning later
            if q is within HR(e) then    // Heuristic 1
                add $e$ to $RSB$    // $e$ is a R$k$SB point of $q$ by Heuristic 1
            else if q is inside DR(e) then
                discard e
            else
                add e to PGSB    // for the next pruning
for each point $p \in PGSB$ do    // Heuristic 2
    if the window based on p and q contains more than k global k-skyband points then
        discard p
    else
        perform the window query (using $H_w$) based on p and q
        if the window query finds less than (k + 1) points then
            add p to RSB    // p is a RkSB point of q by Lemma 2
        else
            discard p
return RSB
```

Proof. This is guaranteed by Lemmas 1 through 4 and Heuristics 1 to 2. □

Lemma 7. *Every data point will be examined, unless one of its ancestor nodes has been pruned.*

Proof. The proof is obvious because all entries that are not pruned by existing global *k*-skyband points (preserved in the set *GSB*) are added to the heap and examined. □

Lemmas 5 and 6 guarantee that, if the proposed algorithms are performed until their termination, they correctly return all reverse *k*-skyband points, i.e., no false hits and no false misses.

5 Experimental Evaluation

In this section, we experimentally evaluate the effectiveness of our developed pruning heuristics and the performance of our proposed algorithms for R*k*SB retrieval, using both real and synthetic datasets. All algorithms were implemented in C++, and all experiments were conducted on an Intel Core 4 Duo 2.8 GHz PC with 4GB RAM.

We employ two real datasets, namely *CarDB* and *NBA*. Specifically, *CarDB* is a 6D dataset, containing 45,311 tuples, which is extracted from *Yahoo! Autos*. In our experiments, we only select two numerical attributes (i.e., *Price* and *Mileage*) of every car. *NBA* includes 15,272 records about 3542 players on 17 attributes, which is available at the website (*www.databasebasketball.com*). Each record provides statistics of a player in a season. Four attributes, including *number of games played* (GP), *total points* (PTS), *total rebounds* (REB), *and total assists* (AST), are considered in our experiments. We also create *Independent* (*IN*) and *Clustered* (*CL*) datasets with dimensionality *dim* in the range [2, 5] and cardinality *N* in the range [40K, 200K]. Specifically, *IN* consists of random points from the unit square. *CL* comprises four clusters, each of them follows a Gaussian distribution. Note that for all datasets, every dimension of the data space is normalized to range [0, 10000]. Each dataset is indexed by an R-tree [1], with a page size of 4096 bytes.

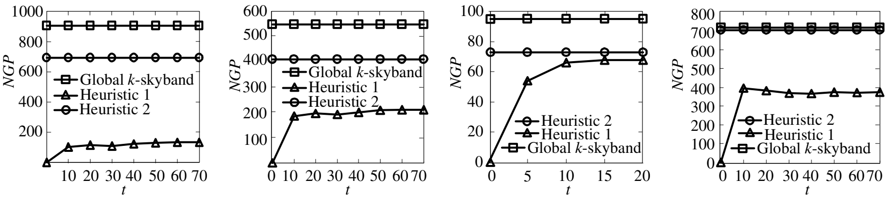
We investigate several factors, involving the number *t* of dynamic skyline points, reverse *k*-skyband thickness *k*, dimensionality *dim*, and cardinality *N*. Note that, in each experiment, only one factor varies, whereas the others are fixed to their default values. The *wall clock time* (i.e., the sum of I/O cost and CPU time, where the I/O cost is computed by charging 10ms for each page access, as with [10]) and *the number of global k-skyband points pruned* (*NGP*) are used as the major performance metrics. Each reported value in the following diagrams is the average of 100 queries whose locations follow the corresponding dataset distribution.

5.1 Effectiveness of Pruning Heuristics

This set of experiments aims at verifying the effectiveness of our proposed pruning heuristics. First, we vary *t* from 10 to 70 (20 for *CarDB*), with *k* fixed at 3 (the median value of Figure 9). The results are shown in Figure 8. Evidently, each heuristic prunes a large number of unqualified global *k*-skyband points, which validates its usefulness. Take Heuristic 2 for *CL* as an example. It saves the detailed examination of 407 out of 544 when *t* = 50. Compared with Heuristic 1, Heuristic 2 has a more powerful pruning capability. This is because, as mentioned in Section 4.3, the global *k*-skyband points are closer to a given query point, compared against the other points, and thus it has a high probability of falling in the window for pruning. Observe that the *NGP* of Heuristic 1 is *zero* when *t* = 0 since there do not exist *DR* and *HR* that can be used to discard unqualified candidate points. Figures 9 to 11 illustrate the prune efficiency of heuristics w.r.t. *k*, *dim*, and *N*, respectively, using both real and synthetic datasets. The diagrams confirm the observations and corresponding explanations of Figure 8.

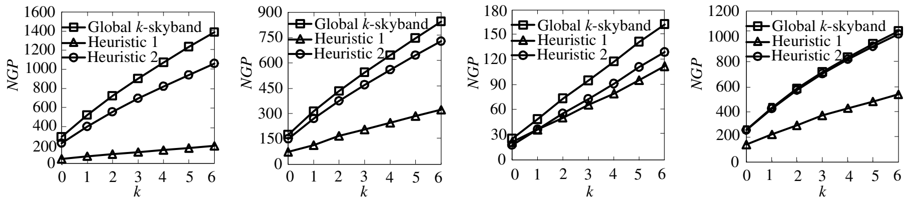
5.2 Results on R*k*SB Queries

The second set of experiments studies the performance of our proposed algorithms (i.e., BR*k*SB, PR*k*SB, and OPR*k*SB) in answering R*k*SB queries. First, we explore the impact of *t* on PR*k*SB and OPR*k*SB, and the results are shown in Figure 12. Note that, since BR*k*SB does not employ dynamic skylines to prune points, it is excluded from this experiment. Clearly, OPR*k*SB outperforms PR*k*SB in all cases because it integrates Heuristics 1 and 2 to prune unqualified candidate points. Furthermore, as *t* grows, the cost of PR*k*SB drops, while that of OPR*k*SB remains almost the same. The reason behind is that the accurate dynamic skylines increase with the growth of *t*, resulting in more unnecessary points pruned and the decrease of the cost.



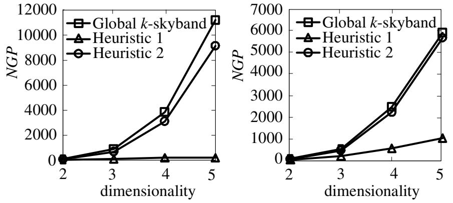
(a) *IN* (3D, 100K) (b) *CL* (3D, 100K) (c) *CarDB* (2D, 45K) (d) *NBA* (4D, 15K)

Fig. 8. Heuristic efficiency vs. *t* (*k*=3)



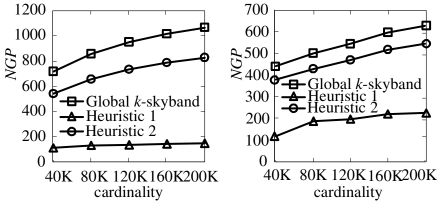
(a) *IN* (3D, 100K) (b) *CL* (3D, 100K) (c) *CarDB* (2D, 45K) (d) *NBA* (4D, 15K)

Fig. 9. Heuristic efficiency vs. *k* (*t*=10 for *CarDB*, and *t*=50 otherwise)



(a) *IN* (100K) (b) *CL* (100K)

Fig. 10. Heuristic Efficiency vs. *dim* (*t*=50, *k*=3)



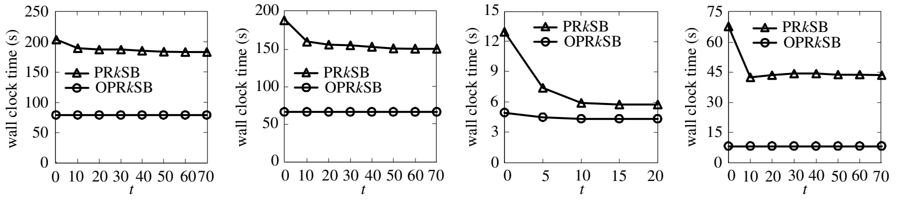
(a) *IN* (3D) (b) *CL* (3D)

Fig. 11. Heuristic Efficiency vs. *N* (*t*=50, *k*=3)

Then, we evaluate the effect of *k* on the algorithms. Figure 13 depicts the cost as a function of *k*. OPRkSB clearly exceeds PR*k*SB and BR*k*SB and the differences ascend fast with *k*. This is because, when *k* increases, more candidate points need to check, which incurs more query cost.

Next, we investigate the influence of *dim* on the algorithms. Towards this, we use the synthetic datasets (i.e., *IN* and *CL*) with *N* = 100K, fix *k* = 3, and vary *dim* from 2 to 5. Figure 14 plots the cost of algorithms w.r.t. *dim*. As expected, the performance of all algorithms degrades with the growth of *dim*. This degradation is due to the poor efficiency of R-trees in high dimensions. However, OPRkSB still performs the best.

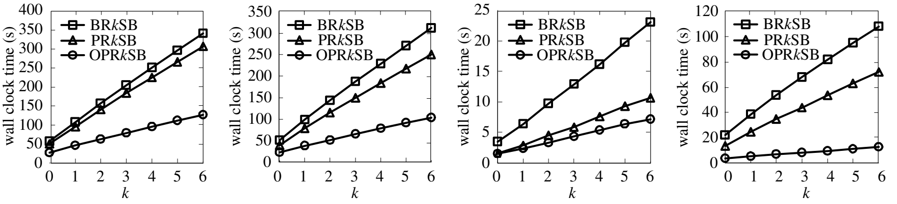
Finally, we inspect the effect of *N* on the algorithms, by fixing *t* = 50, *k* = 3, and employing 3D *IN* and *CL* datasets whose cardinality *N* varies between 40K and 200K. Figure 15 shows the performance of algorithms as a function of *N*. Again, OPRkSB outperforms PRkSB and BRkSB in all cases. Moreover, the cost of algorithms grows as *CN* increases. This is because the size of final query result ascends with *N*.



(a) *IN* (3D, 100K) (b) *CL* (3D, 100K) (c) *CarDB* (2D, 45K) (d) *NBA* (4D, 15K)

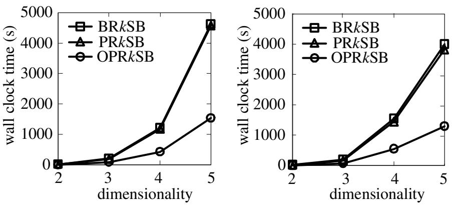
Fig. 12. R*k*SB cost vs. *t* (*k*=3)

In summary, from the above results on real and synthetic datasets, we can conclude that: OPR*k*SB performs the best, followed by PR*k*SB, and BR*k*SB is the worst.



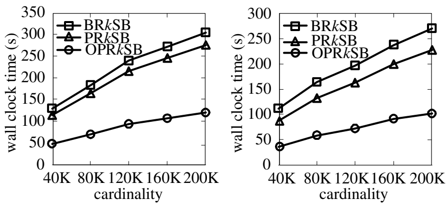
(a) *IN* (3D, 100K) (b) *CL* (3D, 100K) (c) *CarDB* (2D, 45K) (d) *NBA* (4D, 15K)

Fig. 13. R*k*SB cost vs. *k* (*t*=10 for *CarDB*, and *t*=50 otherwise)



(a) *IN* (100K) (b) *CL* (100K)

Fig. 14. R*k*SB cost vs. *dim* (*t*=50, *k*=3)



(a) *IN* (3D) (b) *CL* (3D)

Fig. 15. R*k*SB cost vs. *N* (*t*=50, *k*=3)

6 Conclusions

This paper, for the first time, introduces and solves a new form of reverse skyline queries, namely *reverse k-skyband* (R*k*SB) retrieval where, given a data set *P* and a query point *q*, the goal is to find all the points in *P* whose *dynamic k-skyband* contains *q*. In order to efficiently compute the R*k*SB of an *arbitrary* query point, we first develop a *Branch-bound-based RkSB algorithm* (BR*k*SB), which is an improved customization of the original BBS algorithm; and then propose a *Pre-computation-based RkSB algorithm* (PR*k*SB) using accurate pre-computed dynamic skylines; and finally present an *Optimized PRkSB* (OPR*k*SB) that integrates effective pruning heuristics to prune away unqualified candidate points. Extensive experimental evaluation with both real and synthetic datasets demonstrates that OPR*k*SB achieves orders of magnitude performance gain over alternative solutions.

In the future, we intend to devise more efficient algorithm(s) for answering R*k*SB queries by using the reuse technique. Another interesting direction for future work is to extend our methods to tackle other variants of reverse skyline queries, e.g., constrained reverse skyline query, top-*k* reverse skyline query, etc. Finally, we plan to study R*k*SB queries in metric spaces and over uncertain data, respectively.

Acknowledgements. This work was supported by NSFC 61003049, ZJNSF Y110278, 2011AY1005, the Fundamental Research Funds for the Central Universities under grant 2010QNA5051, the Key Project of Zhejiang Univ. Excellent Young Teacher Fund, and the Returned Scholar Fund for the Personnel Office of Zhejiang Province.

References

1. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In: SIGMOD, pp. 322–331 (1990)
2. Borzsony, S., Kossmann, D., Stocker, K.: The Skyline Operator. In: ICDE, pp. 421–430 (2001)
3. Chen, L., Cui, B., Lu, H.: Constrained Skyline Query Processing against Distributed Data Sites. IEEE Trans. Knowl. Data Eng. 23(2), 204–217 (2011)
4. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with Presorting. In: ICDE, pp. 717–719 (2003)
5. Dellis, E., Seeger, B.: Efficient Computation of Reverse Skyline Queries. In: VLDB, pp. 291–302 (2007)
6. Gao, Y., Chen, G.-C., Chen, L., Chen, C.: Parallelizing Progressive Computation for Skyline Queries in Multi-disk Environment. In: Bressan, S., Küng, J., Wagner, R. (eds.) DEXA 2006. LNCS, vol. 4080, pp. 697–706. Springer, Heidelberg (2006)
7. Godfrey, P., Shipley, R., Gryz, J.: Maximal Vector Computation in Large Data Sets. In: VLDB, pp. 229–240 (2005)
8. Kohler, H., Yang, J., Zhou, X.: Efficient Parallel Skyline Processing using Hyperplane Projections. In: SIGMOD, pp. 85–96 (2011)
9. Kossmann, D., Ramsak, F., Rost, S.: Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In: VLDB, pp. 275–286 (2002)
10. Lian, X., Chen, L.: Reverse Skyline Search in Uncertain Databases. ACM Trans. Database Syst. 35(1), 3 (2010)
11. Lin, X., Zhang, Y., Zhang, W., Cheema, M.A.: Stochastic Skyline Operator. In: ICDE, pp. 721–732 (2011)
12. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive Skyline Computation in Database Systems. ACM Trans. Database Syst. 30(1), 41–82 (2005)
13. Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic Skylines on Uncertain Data. In: VLDB, pp. 15–26 (2007)
14. Prasad, M.D., Deepak, P.: Efficient Reverse Skyline Retrieval with Arbitrary Non-Metric Similarity Measures. In: EDBT, pp. 319–330 (2011)
15. Sacharidis, D., Bouros, P., Sellis, T.K.: Caching Dynamic Skyline Queries. In: Ludäscher, B., Mamoulis, N. (eds.) SSDBM 2008. LNCS, vol. 5069, pp. 455–472. Springer, Heidelberg (2008)
16. Tan, K.-L., Eng, P.-K., Ooi, B.C.: Efficient Progressive Skyline Computation. In: VLDB, pp. 301–310 (2001)
17. Wang, G., Xin, J., Chen, L., Liu, Y.: Energy-Efficient Reverse Skyline Query Processing over Wireless Sensor Networks. IEEE Trans. Knowl. Data Eng. (to appear)
18. Wu, X., Tao, Y., Wong, R.C.-W., Ding, L., Yu, J.X.: Finding the Influence Set through Skylines. In: EDBT, pp. 1030–1041 (2009)
19. Wu, P., Zhang, C., Feng, Y., Zhao, B.Y., Agrawal, D., Abbadi, A.E.: Parallelizing Progressive Skyline Queries for Scalable Distribution. In: EDBT, pp. 112–130 (2006)
20. Zhang, S., Mamoulis, N., Cheung, D.W.: Scalable Skyline Computation Using Object-based Space Partitioning. In: SIGMOD, pp. 483–494 (2009)
21. Zhang, W., Lin, X., Zhang, Y., Wang, W., Yu, J.X.: Probabilistic Skyline Operator over Sliding Windows. In: ICDE, pp. 1060–1071 (2009)
22. Zhu, L., Li, C., Chen, H.: Efficient Computation of Reverse Skyline on Data Stream. In: Proc. Int'l Joint Conf. on Computational Sciences and Optimization, pp. 735–739 (2009)

Co-spatial Searcher: Efficient Tag-Based Collaborative Spatial Search on Geo-social Network

Jinzeng Zhang[1], Xiaofeng Meng[1], Xuan Zhou[1,2], and Dongqi Liu[1]

[1] School of Information, Renmin University of China, Beijing, China
[2] Key Labs of Data Engineering and Knowledge Engineering, Ministry of Education, China
{zajize,xfmeng,xzhou}@ruc.edu.cn

Abstract. The proliferation of geo-social network, such as Foursquare and Facebook Places, enables users to generate location information and its corresponding descriptive tags. Using geo-social networks, users with similar interests can plan for social activities collaboratively. This paper proposes a novel type of query, called Tag-based top-*k* Collaborative Spatial (*TkCoS*) query, for users to make outdoor plans collaboratively. This type of queries aim to retrieve groups of geographic objects that can satisfy a group of users' requirements expressed in tags, while ensuring that the objects be within the minimum spatial distance from the users. To answer *TkCoS* queries efficiently, we introduce a hybrid index structure called Spatial-Tag R-tree (*STR-tree*), which is an extension of the R-tree. Based on *STR-tree*, we propose a query processing algorithm that utilizes both spatial and tag similarity constraints to prune search space and identify desired objects quickly. Moreover, a differential impact factor is adopted to fine-tune the returned results in order to maximize the users' overall satisfaction. Extensive experiments on synthetic and real datatsets validate the efficiency and the scalability of the proposed algorithm.

Keywords: Spatial collaborative search, Tag, Geo-social network, Shadow prefix-tree.

1 Introduction

With the wide application of location-acquisition technologies, such as GPS, Wi-Fi and Social Networks, Geo-Social Network (*GeoSN*) is increasingly being used in our daily lives. Some examples of *GeoSNs* include Google Buzz, Foursquare, Facebook Places, etc. In a *GeoSN*, a variety of spatial objects (e.g.restaurants, hotels, businesses) are marked on the map and annotated with user generated tags. *GeoSN* users can search for interesting spatial objects, and share information about their location and activities. More importantly, users with similar interests can plan for social activities collaboratively, such as going to somewhere for dining and shopping, or taking a cycling tour together. To make such plans, it is essential to identify a group of spatial objects, such as restaurants, shops and parks, which can maximally satisfy the users' needs.

In this paper, we study how to find suitable spatial objects to meet *GeoSN* users' needs in collaborative activity planning. We formulate a new kind of spatial queries called **Tag-based top-*k* Collaborative Spatial (*TkCoS*) Query**, which aims to retrieve top-k groups of objects for meeting users' needs. In essence, the spatial objects returned

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 560–575, 2012.
© Springer-Verlag Berlin Heidelberg 2012

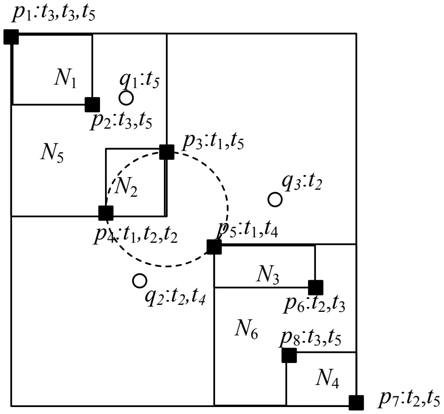


Fig. 1. Example of T*k*CoS query

by a *TkCoS* query should satisfy the following conditions: (1) *they should be annotated with as many tags specified in the query as possible*; (2) *the objects in the result should be as close to one another as possible, such that the maximum diameter of the area covering the objects is minimized*; (3) *the maximum distance between the users' locations and the objects should be minimized.*

Figure 1 illustrates the proposed *TkCoS* query by an example. Points $p_1...p_8$ represent different spatial objects distributed in the region of Los Angeles, each object is annotated with a number of descriptive tags t_i. Suppose three users of a *GeoSN*, Bob, Tom and Mary, plan to find places to meet. They collaboratively submit a *TkCoS* query Q= $\{\langle RowenaAve : ModeratePizza\rangle, \langle MononSt : FreeParking, cinema\rangle, \langle RussellAve : cinema\rangle\}$, where *Rowena Ave($q_1$)* and *Monon St($q_2$)* and *Russell Ave($q_3$)* represent the users' locations, and *Moderate Pizza(t_5), Free parking(t_4), cinema(t_2)* are query tags that express their needs. As shown in Figure 1, the group of objects $\{p_3, p_4, p_5\}$ appears to be the best choice, since it can (1) cover all the user's tags, (2) cover the smallest area, and (3) be near to the users' locations. In contrast, the objects $\{p_2, p_5, p_6\}$ are not suitable. Although they are optimal choices for some individual users (e.g. p_2 for q_1), not all the users' needs are well covered.

TkCoS query can be used in many real world applications. For example, a group of people who want to co-rent a house may have a number of requirements regarding the house's quality, utility, and distance to their working places. These requirements can be met by a single *TkCoS* query. Despite its usefulness,*TkCoS* query poses several challenges to the existing techniques (e.g., [3], [6]) of spatial query processing. Typical spatial keyword queries consider only a single query location (see section 2 for a detailed comparison), and cannot be directly applied to process *TkCoS* queries. A possible adaptation is to use existing (e.g. [6]) methods to execute the sub-queries of a *TkCoS* query separately, and then merge all the results returned by the sub-queries. However, this approach is inefficient. On the one hand , each sub-query has to return a large number of objects to ensure the optimality of the final query results. On the other hand, it will incur high CPU and I/O overheads, as the same data needs to be accessed repeatedly by different queries.

To process *TkCoS* queries efficiently, we devise an efficient hybrid index structure called Spatial-Tag R-tree *(STR)-tree*, which integrates the tag information into a R-tree. To retrieve a group of spatial objects that maximize the users' satisfaction, we propose an algorithm to perform a best-first traversal in the tree. In the algorithm, we employ a *shadow prefix-tree* model to generate candidate sets of search space. An upper bound constraint and a bidirectional constraint are used to prune search space. In addition, we define a differential impact factor to avoid finding the group of objects with covering only a subset of users' requirements. We conduct extensive experiments to evaluate our algorithm using synthetic data sets and real-world data sets. The results demonstrate that the proposed algorithm is efficient and scalable and exhibits superior performance over the brute force method.

The rest of this paper is organized as follows. Section 2 introduces the related work. We formally define the problem of tag-based collaborative spatial search in Section 3. Section 4 introduces the STR-tree. Section 5 introduces our algorithm for processing *TkCoS* queries. Section 6 presents our experimental evaluation. We summarize our work and discuss future work in Section 7.

2 Related Work

In recent years, we have seen an increase in the research dedicated to spatial keyword search. In the query processing of spatial-keyword search, indexing techniques[1],[2], [3],[4],[7] for both text and geographic data are used. Hariharan et al. [1] addressed the problem of spatial keyword queries by utilizing region constraints. Their approach exploits a hybrid index called KR*-tree, which extends R*-tree by augmenting each node with a set of the keywords that appear in the descendants of the node. The query results are the objects located in the query region that are annotated with the query keywords. Felipe et al. [2] proposed a similar kind of query and used IR^2-tree, a combination of R-tree and signature files, to perform query processing. It only contains the information to determine whether a given document contains a query keyword. It is unable to rank the documents based on textual relevance. The work [3] proposes the location-aware top-k text retrieval (L*k*T) query, which takes into account both location proximity and text relevancy. And introduces a hybrid index called IR-tree which integrates R-tree and inverted lists. However, in Web applications, as the number of documents and keywords can be very large, they can result in fat nodes in the IR-trees. The above approaches aim to retrieve only single objects as query results. In contrast, our goal is to find groups of objects such that the objects in a group collectively satisfy the needs of multiple users.

Zhang et al. [4],[5] addressed the problem of m-closest keyword (*m*CK) query. The *m*CK query aims to find the spatially closest tuples which match m user-specified keywords. It utilizes bR*-tree, an integration of R*-tree and bitmap. Each node in the tree is augmented with a keyword MBR to support pruning in the tree traversal. However, as the approach assumes that each object in the result set corresponds to a unique query keyword, it cannot be applied to the cases where multiple constraints are specified on a single object. The work [6] proposes the collective spatial keyword query, aiming to retrieve a group of spatial objects, such that the group's keywords cover the query's keywords and the objects are the nearest ones to the query location. Our *TSkCo* query differs from above approaches in three aspects. First, our query helps multiple users in different locations to search for spatial objects collaboratively. Second, compared to [4][6], our approach aims to find the top-k groups of spatial objects and support partial match of query tags. Third, we exploit vector space model to calculate tag similarity rather than treating all query keywords equally. To summarize, the semantics of *TkCoS* query are different from those of the *m*CK query and collective keyword query.

3 Problem Statement

Let *D* be a set of spatial objects. Each object in *D* is represented by a pair o=$\langle loc,\ t\rangle$, where *loc* represents spatial location information and *t* is a bag of tags for describing the object. In the vector space model of IR[8][9], *t* can be treated as a vector in finite-dimensional space. This vector can be utilized to calculate the similarity between two sets of tags.

A *TkCoS* query can be represented as Q= $\{\langle q_1.loc, q_1.t\rangle, ..., \langle q_m.loc, q_m.t\rangle\}$, where $q_i.loc$ is the *i*th user's location and $q_i.t$ represents a set of tags that describe the users' requirements or preferences. The *TkCoS* query intends to retrieve the top-*k* groups of spatial objects R=$\langle r_1, ..., r_n\rangle$ with the smallest aggregated distance from the users, the minimal spatial coverage and the highest similarity to *Q* measured in descriptive tags.

In order to search for the top-k best object groups from a spatial dataset, we propose a ranking function to measure how well a search result satisfies a *TkCoS* query. The function takes into account both the spatial proximity and the similarity between tag sets. The spatial proximity, denoted by *D(Q,R)*, can be measured by two components. One is the maximum distance between the sub-query locations of *Q* and the result set *R*, denoted by $D_1(Q,\ R)$. The other is the maximum diameter of the area of covering *R*, denoted by $ODiam(R)$. That is to say,

$$Rank(Q,R) = \alpha\frac{D(Q,R)}{\max D} + (1-\alpha)(1-Tr(Q.t,R.t)) \tag{1}$$

$$D(Q,R) = \beta D_1(Q,R) + (1-\beta)ODiam(R) \tag{2}$$

$$D_1(Q,R) = \max_{q_i \in Q}(\sum_{j=1}^{n}(dist(q_i,r_j))) \tag{3}$$

In Formula (1), $Tr(Q.t,R.t)$ denotes the tag similarity between *Q.t* and *R.t*. $\max D$ denotes the maximal distance between any two objects in *D*. It is used as a normalization factor. In Formula (3), $dist(q_i,\ r_j)$ denotes the Euclidean distance between an object $r_j \in R$ and a sub-query's location $q_i \in Q$. The parameters $\alpha,\beta \in (0,1)$ are used to adjust the tradeoff between the factors. To measure the maximal diameter of the area covering *R*, $ODiam(R)$, we give the following definition.

Definition 1. *Given a set of spatial objects R=*$\langle r_1, ..., r_n\rangle$*, the diameter of R, denoted as ODiam.*

$$ODiam(R) = \max_{r_i \in R, r_j \in R}(dist(r_i,r_j)) \tag{4}$$

where dist(r_i*,*r_j*) measures the Euclidean distance between the two objects* r_i *and* r_j*.*

Compared to a normal document, a tag set usually consists of a much smaller number of terms. Therefore, a direct application of a traditional IR model to measure the tag similarity $Tr(Q.t,R.t)$ in Formula (1) can lead to inadequate results. In this paper, we adopt the method proposed in [10] as our similarity metric, which is defined as follows:

$$Tr(Q.t,R.t) = \sum_{q_i \in Q, r_j \in R} (simt(q_i.t, r_j.t)) \tag{5}$$

$$simt(q_i.t, r_j.t) = \frac{(q_i.t)C(r_j.t)^T}{\sqrt{(q_i.t)C(q_i.t)^T}\sqrt{(r_j.t)C(r_j.t)^T}} \tag{6}$$

In Formula (6), C is a tag similarity matrix, which can be represented by $C{=}(c_{i,j})_{n\times n}$, where n is the number of distinct tags, and $c_{i,j}$ is the similarity value between two tags t_i and t_j.

Finally, the goal of a *TkCoS* query is to find groups of spatial objects with the smallest $Rank(Q,R)$. Our problem can be defined as follows.

Definition 2. *(TkCoS Retrieval). Given a dataset D and a TkCoS query* $Q{=}\{\langle q_1.loc, q_1.t\rangle, \ldots, \langle q_m.loc, q_m.t\rangle\}$*, find k groups of objects* $\{R_1, R_2, \ldots, R_k\}$ *(*$R_i{=}\{r_{i1}, r_{i2}, \ldots, r_{in}\}$*), such that there does not exist* $R' \notin \{R_1,R_2,...,R_k\}$ *that satisfies* $Rank(Q,R') < Rank(Q,R_i)$ *where* $R_i \in \{R_1,R_2,...,R_k\}$*.*

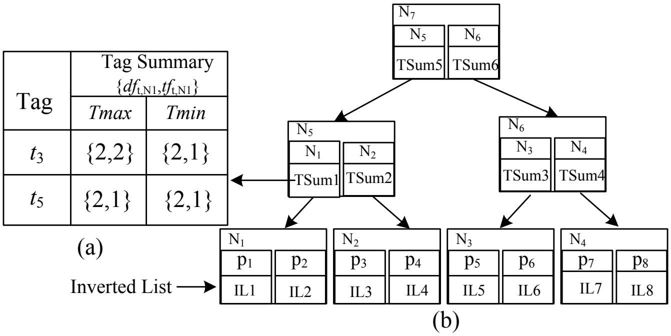
4 STR-Tree: A Refined Hybrid Indexing Mechanism

To answer *TkCoS* queries efficiently, we introduce an efficient hybrid index structure called Spatial-Tag R-tree (*STR-tree*), which is an extension of IR-tree [3] and the original R-tree [11]. It clusters spatially close and semantically relevant objects together and stores the tag information in the nodes of the R-tree [11].

In the *STR-tree*, a leaf node includes entries in the form $(optr, loc, oti)$, where *optr* is a pointer to an object in D, *oti* represents the tag information of an object, which is indexed by inverted lists [12]. A intermediate node contains these entries in the form $(Nptr, MBR, Ntsum)$, where *Ntsum* represents the tag summary information of its child nodes referred by *Nptrs*. The *Ntsum* includes two parts: tag maximum information *Tmax* and tag minimum information *Tmin*. Note that each tag in the inverted lists is associated with a tag frequency (*tf*) and the number of objects containing the tag (*df*). To minimize storage overhead, for each tag, the *Tmax* (resp. *Tmin*) of each non-leaf node N_i stores only the *df* and the maximum (resp. minimum) *tf* among all the child nodes rooted at N_i. This maximum (resp. minimum) *tf* provides an upper (resp. lower) bound of the tag similarity between a query and the nodes in the subtree rooted at N_i.

Fig.2 gives an example of *STR-tree* for the spatial objects in Figure 1. Fig. 2(a) shows the *Tmax* and *Tmin* information of the non-leaf node N_1. In Fig. 2(b), the objects p_1 and p_2 are grouped into the node N_1. Likewise, p_3 and p_4 are grouped into N_2. These two non-leaf nodes form a intermediate higher-level node N_5, and so on.

The construction of a *STR-tree* is conducted through a sequence of insert operations, which are a well studied operation in the original R-tree. The only difference is that it needs to update the tag maximal and tag minimal information. Similarly, the update and delete operations of *STR-tree* are simple extensions of those of R-tree too.



Tag	Tag Summary $\{df_{t,N1}, tf_{t,N1}\}$	
	Tmax	*Tmin*
t_3	{2,2}	{2,1}
t_5	{2,1}	{2,1}

(a)

(b)

Fig. 2. STR-tree indexing structure

5 Processing *TkCoS* Queries

Comparing with other types of spatial keyword queries, a *TkCoS* query is a collaborative query composed of multiple locations associated with multiple tags. A brute force approach is to process each sub-query q_i in Q independently, and merge all the results returned by the sub-queries. Obviously, this approach will lead to high processing cost. First, the same node will be accessed repeatedly in different sub-queries. Second, we need to keep the result set of each sub-query sufficiently large, to ensure the merged results contain the top-k.

In this section, we present a more efficient algorithm to answer *TkCoS* queries. Our idea is to perform a best-first search on the *STR-tree*. When performing the search, we maintain a ranked list of ***candidate node sets***, where each set is a set of the nodes in the *STR-tree* that can potentially contain a top-k result. In each step of the search, we pick the candidate node set with the minimal rank score, and start from its node to traverse the *STR-tree*. Then we use the new nodes encountered in the traversal to form new candidate node sets, and insert them into the ranked list. The candidate node sets are ranked based on the minimum possible score of the results it could contain. During the best-first search, we utilize several pruning strategies to truncate the irrelevant nodes in the *STR-tree*, such that a significant part of the tree can be skipped.

5.1 Query Algorithm

The efficiency of the query algorithm depends on how we evaluate the fitness of each candidate node set. It determines how fast we can reach the bottom of the *STR-tree* and how many irrelevant nodes can be pruned during the search process. To evaluate each candidate node set, we utilize two metrics, that is, the lower bound and the upper bound of the possible scores (defined in Formula (1)) of the results this candidate node set contains. Let *NS* be a candidate node set, and let *Q* be the query, we denote the lower bound and upper bound by *MinRank(Q,NS)* and *MaxRank(Q,NS)* respectively.

Obviously, the lower bound *MinRank(Q,NS)* is used to rank the candidate node sets encountered during the search and prune the paths of the search space in the hybrid index, so as to guarantee that the top-k results returned sequentially.We compute *MinRank(Q,NS)* as follows:

Definition 3. *Given a TkCoS query Q and a node set NS, the minimal possible score of the results in NS w.r.t Q (MinRank) is:*

$$MinRank(Q,NS) = \alpha \frac{MIND_{\varepsilon}(Q,NS)}{\max D} + (1-\alpha)(1-maxTr(Q.t,NS.u)) \quad (7)$$

$$MIND_{\varepsilon}(Q,NS)) = \beta \max_{n_i \in NS} minDist(Q,n_i) + (1-\beta)(\max_{n_i,n_j \in NS} minDist(n_i,n_j)) \quad (8)$$

where $MIND_{\varepsilon}$(Q, NS) is the minimal spatial proximity between Q andNS, $\max_{n_i \in NS}$ *minDist(Q,n_i) is the minimal Euclidian distance between Q and NS,* $\max_{n_i,n_j \in NS}$ *minDist (n_i,n_j) is the minimal diameter of NS, $maxTR(Q.t,NS.u)$ is the maximal tag similarity of Q and NS, and α, β and maxD are the same as those in Forumla (1)and (2).*

Lemma 1. ***MinRank(Q, NS)*** *satisfies the following property.*

$$\forall_{os \in Oset} Rank(Q, os) \geq MinRank(Q, NS) \quad (9)$$

where Oset is the spatial object set contained in node NS, os is any subset of Oset.

Proof. First, according to Formula (3), we have *D_1(Q, os) ≥ minDist(Q, NS)*. Second, according to *Definition 1*, the diameter of a node set is the maximal distance of any pair of its nodes. Thus, we have: ODima(os) > $\max_{n_i,n_j \in NS}$ minDist(n_i,n_j). Third, since *maxTr(Q.t, NS.u)* is the upper bound of tag similarity between *Q* and *NS*, we can *Tr(Q.t, o.t) ≤ maxTr(Q.t, NS.u)*. We can derive *Rank(Q, os)≥ MinRank(Q, NS)* for any *os*. □

Lemma1 proves that *MinRank(Q, NS)* is a true lower bound. Therefore, if we traverse a *STR-tree* in the ascending order of *MinRank(Q, NS)*, we guarantee to find the top-*k* results of *Q*.

The upper bound *MaxRank(Q,NS)* is used to prune the inappropriate candidate node sets as early as possible in search processing. It is calculated as follows.

Definition 4. *Given a TkCoS query Q and a node set NS, the maximum possible score of the results in NS w.r.t Q (MaxRank) is:*

$$MaxRank(Q,NS) = \alpha \frac{MAXD_{\varepsilon}(Q,NS)}{\max D} + (1-\alpha)(1-minTr(Q.t,NS.l)) \quad (10)$$

$$MAXD_{\varepsilon}(Q,NS)) = \beta \max_{n_i \in NS} maxDist(Q,n_i) + (1-\beta)(\max_{n_i,n_j \in NS} maxDist(n_i,n_j)) \quad (11)$$

where $MAXD_{\varepsilon}$(Q, NS) is the maximal spatial proximity between Q andNS, $\max_{n_i \in NS}$ *maxDist(Q,n_i) is the maximal Euclidian distance between Q and NS, $minTR(Q.t,NS.l)$ is the minimal tag similarity between Q and NS, and* $\max_{n_i,n_j \in NS}$ *maxDist(n_i,n_j) is the maximal diameter of NS, denoted as* ***maxDima****.*

Lemma 2. ***MaxRank(Q, NS)*** *satisfies the following property.*

$$\forall_{os \in Oset} Rank(Q, os) \leq MaxRank(Q, NS) \tag{12}$$

where Oset is a spatial object set contained in node NS, os is any subset of Oset.

Proof. This lemma can be proved in a similar way as Lemma 1. □

Lemma 3 (Upper Bound Constraint). *Given a TkCoS query Q and a candidate node set NS, Let CNS_k be the kth candidate node set based on the ascending order of MaxRank in the maintained list. The node set NS can be disregarded during traversing the STR-tree if MinRank(Q,NS)> MaxRank(Q,CNS_k).*

Proof. Denoted by *os* the object set enclosed in the node set *NS*. According to Lemma 1, we have: *Rank(Q,os)* $\geq$ *MinRank(Q,NS)*. As *MinRank(Q,NS)> MaxRank (Q,CNS_k)*, we can derive that *Rank(Q,os)* $\geq$ *MaxRank(Q,CNS_k)*. Therefore, *NS* cannot contain any top-k results. □

Lemma 2 and 3 proves that *MaxRank(Q,CNS_k)* is a upper bound(denoted as ***uppC***) of the candidate node sets. Using Lemma 3, we can prune the candidate node set that cannot possibly contain top-k.

In the query processing, apart from considering the ranking function in Formula 1, we should also care the satisfaction degree of each individual user. The objects returned only covering a handful of users' needs should be eliminated from the results. In our work, we adopt *Bayes theory* to define *Contribution Degree* of each sub-query q_i in Q.

Definition 5. ***(Contribution Degree.)*** *Given a TkCoS Q=* $\{\langle q_1.loc, q_1.t\rangle, ..., \langle q_m.loc, q_m.t\rangle\}$ *and a node set NS, let* $q_1, q_2, ..., q_m$ *be a partition of Q. Contribution Degree of* q_i *can be defined as follows.*

$$P(q_i|NS) = \frac{P(NS|q_i)P(NS|q_i)}{P(NS)} \tag{13}$$

where $P(q_i)$*=simt(*q_i*.t, NS.t),* $P(NS|q_i)$*=*$|q_i.t \cap NS.t|/|NS.t|$*, and P(NS) =* $|Q.t \cap NS.t|/|Q.t|$.

According to Formula (13), when the contribution degree of each sub-query respectively infinitely tend to the proportion of $|\ q_i.t\ |$ in $|\ Q.t\ |$, the users can be maximally satisfied. Therefore, we use the difference between the contribution degree of and q_i and $\frac{|q_i.t|}{|Q.t|}$ to measure degree of satisfaction. We call this difference *Differential Impact Factor.*

Definition 6. ***(Differential Impact Factor)*** *Given a TkCoS Q=* $\{\langle q_1.loc, q_1.t\rangle, ..., \langle q_m.loc, q_m.t\rangle\}$ *and a node set NS, the differential impact factor* δ *is defined as follows.*

$$\delta_{NS} = \frac{\sqrt{\sum_{i=1}^{m}(P(q_i \mid NS) - \frac{|q_i.t|}{|Q.t|})^2}}{\sqrt{m} max(P(q_i \mid NS) - \frac{|q_i.t|}{|Q.t|})} \tag{14}$$

To take the users' satisfaction degree into account, we use δ_{NS} to modify the lower bound *MinRank(Q, NS)*. Note that the value of δ_{NS} is smaller, the overall satisfaction is better. If $\delta_{NS} \in (0,1)$ is too small, the searching order can be changed obviously. To be specific, we apply $e^{\delta_{NS}}$*MinRank(Q, NS)* to rank the candidate node sets.

Lemma 4 (Bi-directional Constraint). *Given a node set NS and the current node set CNS with the smallest MinRank score, if* $e^{\delta_{NS}}$*MinRank(Q, NS)* $\geq$ $e^{\delta_{CNS}}$*MinRank(Q, CNS), then the node set NS is pruned.*

Proof. Obvious from Lemma 1 and definition 3 and 5. □

In order to find top-*k* groups of spatial objects, *STR-tree* is traversed from the root node following the best-first traversal strategy. The pseudocode is shown in Algorithm 1. Let a min-priority queue *U* keep track of candidate node sets *E* with e^{δ_E}*MinRank(Q, E)* , while an ordered link list *LL* store the same nodes in *U* associated with *MaxRank(Q,E)* and the *maxDima(E)* in the ascending order of *MaxRank*. The process iteratively checks the first entry *E* in *U*(line 4-23). If *E* contains only spatial objects, it is returned as a top-k result. Otherwise, If *E* is a intermediate node set, we invoke Algorithm 2 to the children of the nodes in *E*, and compose them into new candidate node sets S_{nl} (line 11).

```
Algorithm 1: COSS(Q, STR-tree, k)
Input: Q: a TkCoS query;
       STR-tree: a hybrid indexing;
Output: The top-k groups of objects satisfying Q;
1:  U ← new min-priority queue; LL ← new a ordered link list;
2:  U.Enqueue(STR-tree.root, 0); LL.Insert(STR-tree.root,∞,∞);
3:  uppC ← ∞; uppDima ← ∞;
4:  while U is not empty do
5:    E← U.Dequeue(); LL.Delete(E);
6:    uppC ← LL[k]; uppDima ← max(maxDima(LL[1..k]));
7:    if E is a group of objects then
8:      R← R ∪{E};
9:      if | R | =K then goto 25;
10:   else if E is a intermediate nodeset then
11:       $S_{nl}$← GenCSet(E, uppC, uppDima);
12:       for each nodeset NS in $S_{nl}$ do
13:           if | U.length| < k or MinRank(NS, Q) < uppC then
14:               U.Enqueue(NS, $e^{\delta_{NS}}$MinRank(NS, Q));
15:               LL.Insert(NS, MaxRank(NS, Q),maxDima(NS));
16:               uppC ← LL[k]; uppDima ← max(maxDima(LL[1..k]));
17:     else if E contains leaf nodes then
18:       $S_l$← GenCSet(E, uppC, uppDima);
19:       for each objectset os in $S_l$ do
20:           if MinRank(os, Q) < uppC then
21:               U.Enqueue(os, $e^{\delta_{os}}$Rank(Q, os));
22:               LL.Insert(os, MaxRank(os, Q),maxDima(os));
23:                uppC ← LL[k]; uppDima ← max(maxDima(LL[1..k]));
24:  return R
```

Then, we consider each of the new node sets. If the node set *NS* in S_{nl} does not satisfy the condition in Lemma 3, it is enqueued to *U* together with $e^{\delta_{NS}}$*MinRank(Q, NS)* and is inserted *LL* with *MaxRank(Q,NS)* and *maxDima(NS)*. Otherwise, *NS* will be discarded, because it cannot contain any top-k. Whenever *LL* changes, we need to update *uppC*, which represents *k*-th smallest *MaxRank(Q,NS)* in *U*, and *uppDima* that is the maximal *maxDima* of top-*k* element in *LL*(line 16). Likewise, if E is a leaf node set, we process E in the same way to the non-leaf nodes (line 17-23). The algorithm repeats the above procedure. Once *R* contains *k* groups of objects or no more groups of objects can be found, the algorithm terminates and outputs *R*.

5.2 Generating Candidate Node Sets of Search Space

During each step of the best-first search algorithm, it needs to expand the nodes in a candidate node set, and use their child nodes to generate more concrete candidate node sets. An efficient generation approach is essential to ensure the efficiency of the top-k algorithm. However, if we exhaustively enumerate all the subsets, it could incur high computing overhead, as the number of subsets grows exponentially with the number of child nodes. In order to reduce the cost of I/O and computation, we need to filter out irrelevant node sets as early as possible. We exploit the *apriori* property among the set and its superset to reduce search space in generating candidate node sets. By using the

Algorithm 2: GenCset(*S*, *uppC*, *uppDima*)

Input: *S*: a set of spatial nodes;

Output: A list of candidate spatial node sets *SList*;

```
1:  $Slist \leftarrow \emptyset$; $T \leftarrow$ SPF-tree($S$,$uppDima$);
2:  for each node $n_i$ in $S$ do
3:      for each childnode $cn_i$ in $n_i$ do
4:          if $cn_i.t \cap Q.t \neq \Phi$ and $\alpha MIND_{\varepsilon}(cn_i,Q) < uppC$ then
5:              $I_1 \leftarrow I_1 \cup cn_i$;
6:  for $k$ $form$ 2 $to$ |Q.t| do
7:      $NN_k \leftarrow$ GenNeighbor($I_{k-1}$, $T$);
8:      for each nodeset $NS$ in $NN_k$ do
9:          if $\alpha MinDist_{\varepsilon}(NS,Q) < uppC$ then
10:             $I_k \leftarrow I_k \cup NS$;
11:     $L \leftarrow \cup_k I_k$;
12: for each nodeset $NS \in L$ do
13:     if ($MinRank(NS, Q) < uppC$) then
14:         add $NS$ to $SList$;
15: return $SList$;
Procedure GenNeighbor($I_{k-1}$, $SPF\text{-}tree$)
16: for each nodeset $l$ in $I_{k-1}$ do
17:     PreOrderTraverse(SPF-tree);
18:     for each node $n_i$ in $l$ do
19:         $CS_{ni} \leftarrow$ Get-Childnodeset($n_i$);
20:         $CommonCS \leftarrow \cap_{ni} CS_{ni}$;
21:     for each node $CN$ in $CommonCS$ do
22:         $C \leftarrow Merge(n_i,CN)$; add $C$ to $NN_k$;
23: return $NN_k$;
```

upper bound of candidate node sets *uppC* and the upper bound of the diameter *uppDima* introduced in section 5.1, we devise two pruning mechanisms to filter out the candidate node sets that cannot possibly contain any top-k result.

Lemma 5. *Given a TkCoS query Q and a node set NS, if* $\alpha MIND_\varepsilon(NS,Q) > uppC$*, then the node set NS and all its supersets cannot contain any top-k result.*

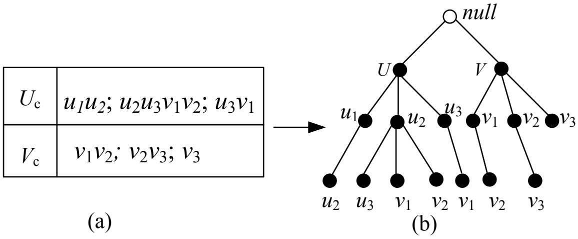
Proof. According to definition 3, we have $MinRank(Q,NS) = \alpha\frac{MIND_\varepsilon(Q,NS)}{\max D} + (1-\alpha)(1-maxTr(Q.t,NS.u)$. On the one hand, the minimal spatial proximity of a superset of *NS* is larger than $MIND_\varepsilon(Q,NS)$. On the other hand, the tag similarity *maxTr(NS,Q)* is in the range between 0 and 1. If we set *maxTr(NS,Q)* to 1, then $\alpha MIND_\varepsilon(Q,NS)$ is the lower bound of the *MinRank* of all its superset. Therefore, when $\alpha MIND_\varepsilon(NS,Q)$ $>$ *uppC* holds, any superset of *NS* has larger *MinRank* score than the scores of the current top-k candidate node sets (because *uppC* is a upper bound). Thus *NS* and all its supersets cannot contain any top-k result. □

By applying Lemma 5 to the generation of the candidate node sets, the node sets that cannot affect the query results can be discarded as early as possible. We call the node set that does not satisfy the condition in Lemma 5 *Relevant Node Set*, denoted as ***I***. Besides, we still utilize the the upper bound of diameter *uppDima* for pruning.

Lemma 6. *Given a node set* $NS=\langle N_1,\ldots,N_k\rangle$*, if the diameter of NS is larger than uppDima where uppDima is the maximal maxDima of top-k element in link list, then NS and its superset can be pruned.*

Proof. According to definition 1, if the diameter of *NS* is larger than *uppDima*, then there exists two nodes $N_i,N_j \in N$ with $minDist(N_i,N_j) \geq uppDima$. Any superset of *NS* must contain N_i,N_j and its diameter exceeds the *uppDima*. Thus *NS* and its superset does not provide a query result with a diameter less than *uppDima*. □

Lemma 6 says that the diameter of the candidate node set can not exceed *uppDima*. In generating candidate node sets, we only care these node sets with neighbor relationship that the distance of any two nodes is less than *uppDima*. Once any two nodes in *NS* satisfy neighbor relationship, we call it a *Neighbor Node Set*, denoted as ***NN***.



U_c	u_1u_2; $u_2u_3v_1v_2$; u_3v_1
V_c	v_1v_2; v_2v_3; v_3

(a) (b)

Fig. 3. The construction of *shadow prefix-tree*

In the sequel, we proceed to propose the strategy of generating candidate node sets. A good candidate generation method keeps the aprior properties, as well as avoids amounts of join operations. Based on this principle and lemma 6, we propose a shadow prefix-tree model that materializes the neighbor relationship between childnodes of *NS*. The number of subtree is determined by the number of nodes in *NS* while each branch in subtree records the neighbor relationship of childnodes of *NS*. Figure4 illustrate an example of shadow prefix-tree about node set $(U(u_1,u_2,u_3),V(v_1,v_2,v_3))$. we can find the neighbor node set by traversing the shadow prefix-tree according to lemma 7.

Lemma 7. ***(Neighbor Node Set Generation)*** *Given a relevant node set* $I_{k-1}=\{n_1,n_2,...,n_{k-1}\}$*, if a node* n_k *is contained in the intersection of child nodes of each node in* I_{k-1}*,* $NN_k=\{n_1,n_2,...,n_k\}$ *is a size k neighbor node set.*

Proof. Each node n_i in $I_{k-1}=\{n_1,n_2,...,n_{k-1}\}$ has neighbor relationship with other node of I_{k-1}. If a node n_k is the child node of $\{n_1,n_2,...,n_{k-1}\}$, then indicates that n_k has a neighbor relationship with all nodes in I_{k-1}. In addition, the neighbor relationship is symmetric. So $NN_k=\{n_1,n_2,...,n_k\}$ is a size-k neighbor node set. □

Algorithm 2 shows the process of generating candidate node sets. In this algorithm, NN_k represents size-k neighbor node sets, I_k is the size-k relevant node sets. The shadow prefix-tree T is firstly built by utilizing S and the upper bound of diameter *uppDima* (line 1). Then, we invoke procedure GenNeighbor (line 16-23) to generate the neighbor node sets NN_k based on T and node set I_{k-1}. I_k can be obtained by filtering the node sets in NN_k that satisfy the condition of lemma 5 (line 9-10). Finally, all of node sets in I_k are appended to list L(line 11). After all node sets L are found, we check each node set in L to see whether its *MinRank* score is less than *uppC* (line 12-14). Those that cannot qualify the conditions are eliminated. On contrary, we do not check this constraint in the process of node set I_k since if a node set does not contain query tags, it can still combine other nodes to covering the missing tags. As long as it is relevant to the tag of query, we keep it in list L of node set I_k.

6 Experiments

This section presents an extensive experimental evaluation of the proposed method for *TkCoS* queries using synthetic and real datesets.

6.1 Experimental Setting

We used two **Baseline Algorithms** to compare with our proposed algorithm **COSS**. **Baseline1 First separate last union (FSLU).** In FSLU, we process each subquery q_i separately using an existing spatial keyword query processing method proposed in [2]. We utilize *STR-tree* to retrieve the object set with smallest rank score for each q_i. Then we merge the results of the sub-queries to obtain the final top-k.

Baseline2 Centroid-based Iterative Search Algorithm (CISA). In CISA, we use an aggregation centroid c to substitute for a *TkCoS* query Q. The tag information of the centroid c can be considered as the tags combination of each subquery in Q. The query processing iteratively utilizes the methods of [2] to retrieve the group of objects that cover all tags and are nearest to centroid c. It firstly finds the object with the smallest distance that matches a part of tags. The uncovered tags together with the location of centroid *c* form a new query. This process terminates until the tags are either matched or skipped (for there is no matching object).

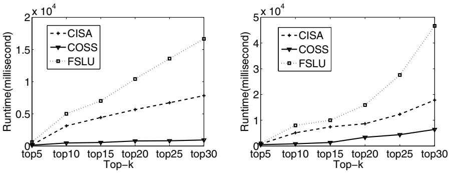
Datasets and Queries. Our experiments used two datasets, whose properties are summarized in Table 1.The real data set DATA1 is obtained from the online web data resource PocketGPSWorld [13] that consists of points of interest locations in United Kingdom. DATA2 is a synthetic dataset generated to simulate a geo-social application. We extracted 3000,000 tags from del.ic.ious [14] and combined the tags with the real spatial dataset about California's streets to generate DATA2. We generated 5 query sets for DATA1, each containing 2,4,8,16,32 tags respectively. Similarly, we generated 5 query sets for DATA2. Each of the query sets comprises of 50 queries and each query is randomly generated. Based on the query sets, we generate 6 group query sets for each dataset. The number of the sub-queries in each group query ranges from 2 to12.

We implement all the algorithms using VC++. In all the experiments, the index structures were disk-resident and the page size was fixed at 4KB. In an index, the number of children in each node is determined solely by the size of a page. All our experiments were executed on a Windows platform with an Intel(R) Core(TM)2 Duo CPU of T7500 @ 2.66GHZ and 4GB RAM.

Table 1. Shows more details of the two datasets

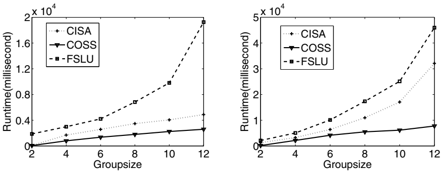
Dataset	Total # of objects	Total # of unique tags	Total # of tags
DATA1	125,313	47,672	877,191
DATA2	2,249,727	289,175	8,998,908

6.2 Performance Evaluation



(a) DATA1 (b) DATA2

Fig. 4. Effect of the *k* value



(a) DATA1 (b) DATA2

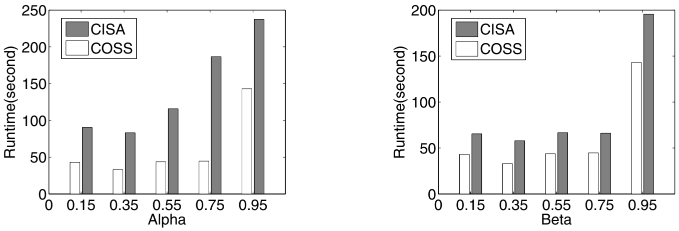
Fig. 5. Effect of the group size

We compare our algorithm COSS against FSLU and CISA in answering *TkCoS* queries. The running time is used as our performance metric. We conduct four sets of experiment in total.

Effect of k. In this set of experiments, we evaluate the performance of the three algorithms with a varying *k*. As shown in Fig.4(a) and 4(b), the COSS method notably outperforms FLSU and CISA for all values of k. Meanwhile, CISA performs better than FSLU. This is mainly because COSS can prune irrelevant nodes more effectively than the other two methods. As expected, the running time of all the approaches increases with increasing k.

Effect of the Group Size of Query. The objective of our second set of experiments is to study the efficiency of the three algorithms in dealing with different sizes of query groups. The results on DATA1 and DATA2 are shown in Fig.5(a) and 5(b). We can see that COSS significantly outperforms both FSLU and CISA. For all the approaches, the query running time increases as group size grows. This is because when increasing group size, it takes more time to process the increasing number of entries in the hybrid index. Nevertheless, the growth rate for COSS is much smaller than the others.

Effect of α and β. Fig.6(a) shows the performance of CISA and COSS on DATA1 with respect to different α. It is clear that COSS significantly outperforms CISA for all the values of α. Recall that α can adjust the importance between the spatial proximity and the tag similarity. A larger α means that the spatial distance is more important, while a smaller α means that the tag information is more important. We notice that the running time increases as α increases. This is mainly because spatial proximity is normally less selective in pruning irrelevant results. The impact of the parameter β on the performance of CISA and COSS algorithm is shown in Fig.6(b). As mentioned earlier, β is introduced to balance the importance of the distance between query Q and results R and that of the covering area of R. We obverse that COSS also outperforms CISA slightly in most cases.

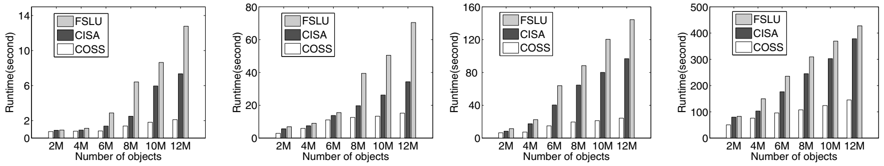


(a) α

(b) β

Fig. 6. Effect of α and β

Scalability in Terms of Dataset Size. In order to simulate the real geo-social networking in which the number of objects and tags continuously increasing, our final set of experiments is conducted to evaluate the scalability of three algorithms by varying the number of objects. We increase the size of the synthetic dataset steadily from 2 million to 12 million. Fig.7 shows the running time of the algorithms as the data size increases. When the group size is small, the CISA and COSS shows the similar rate of increase in running time. As group size grows, CISA's running time increases more dramatically. We can also see that all the algorithms scale smoothly when the number of objects is not greater than 4 million. However, the performance of FLSU and CISA declines quickly when the dataset size is above 4 million. On the contrary, our COSS method scales well even with large dataset size.



(a) groupsize=2 (b) groupsize=4 (c) groupsize=6 (d) groupsize=8

Fig. 7. Scalability in terms of dataset size

7 Conclusions

In this paper, we study the problem of tag-based top-k collaborative spatial (*TkCoS*) query , which aims to find groups of objects with the smallest rank score and the highest satisfaction degree for multiple users. We present an efficient query processing algorithm that is based a hybrid index, STR-tree and employ upper bound constraint and bi-directional constraint to prune irrelevant subtree. This algorithm tackles the key challenge by building a shadow prefix tree model to generate candidate search space. Our experimental evaluation shows that the proposed algorithm is efficient and scalable and superior performance compared with baseline method. Our results can be used as a value-added service in today's social networking websites or geo-based applications.

Acknowledgments. This research was partially supported by the grants from the Natural Science Foundation of China (No. 61070055, 60833005, 91024032, 91124001), the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University of China (No. 12XNLJ01, 11XNL010, 10XNI018), National Science and Technology Major Project (No. 2010ZX01042-002-003).

References

1. Hariharan, R., Hore, B., Li, C., Mehrotra, S.: Processing spatial keyword (sk) queries in geographic information retrieval systems. In: 19th IEEE International Conference on Scientific and Statistical Database Management, pp. 161–170. IEEE Press, Washington (2007)
2. Felipe, I.D., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: 24th IEEE International Conference on Data Engineering, pp. 656–665. IEEE Press, Washington (2008)
3. Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. J. Proc. of VLDB Endowment 2(1), 337–348 (2009)
4. Zhang, D.X., Chee, Y.M., Mondal, M., Tung, A.K., Kitsuregawa, M.: Keyword search in spatial databases: Towards searching by document. In: 25th IEEE International Conference on Data Engineering, pp. 688–699. IEEE Press, Washington (2009)
5. Zhang, D.X., Ooi, B.C., Tung, A.K.H.: Locating mapped resources in web 2.0. In: 26th IEEE International Conference on Data Engineering, pp. 521–532. IEEE Press, Washington (2010)
6. Cao, X., Cong, G., Jensen, C.S.: Collective spatial keyword querying. In: 31th ACM International Conference on Management of Data, pp. 373–384. ACM Press, New York (2011)
7. Khodaei, A., Shahabi, C., Li, C.: Hybrid Indexing and Seamless Ranking of Spatial and Textual Features of Web Documents. In: Bringas, P.G., Hameurlain, A., Quirchmayr, G. (eds.) DEXA 2010. LNCS Part I, vol. 6261, pp. 450–466. Springer, Heidelberg (2010)

8. Wong, S.K.M., Ziarko, W., Raghavan, V.V.: On modeling of information retrieval concepts in vector space. ACM Transaction on Database System 12(2), 299–321 (1987)
9. Anh, V.N., de Kretster, O., Moffat, A.: Vector space ranking with effective early termination. In: ACM 24th International Conference on Research and Development in Information Retrieval, pp. 35–42. ACM Press, New York (2001)
10. Park, J., Choi, B.C., Kim, K.: A vector space approach to tag cloud similarity ranking. J. Information Processing Letters 110(12-13), 489–496 (2010)
11. Guttman, A.: R-Trees: A dynamic index structure for spatial searching. In: 4th ACM International Conference on Management of Data, pp. 47–57. ACM Press, New York (1984)
12. Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Computing Surveys 38(2), 6 (2006)
13. PocketGPSWorld, `http://www.pocketgpsworld.com/modules.php?name=POIs`
14. Delicious, `http://www.delicious.com/`

Traffic Aware Route Planning in Dynamic Road Networks

Jiajie Xu[1], Limin Guo[1,3], Zhiming Ding[1], Xiling Sun[1,3], and Chengfei Liu[2]

[1] Intsitute of Software, Chinese Academy of Sciences, China
{jiajie,limin,zhiming,Xiling}@nfs.iscas.ac.cn
[2] Faculty of ICT, Swinburne University of Technology, Australia
cliu@swin.edu.au
[3] Graduate University of Chinese Academy of Sciences, China

Abstract. The current widespread use of GPS navigations and trip planning on web has aroused great interests in fast and scalable path query processing. Recent research has mainly focused on static route optimisation where the traffic network is assumed to be stable. However in most cases, route planning is in presence of frequent updates to the traffic graph due to the dynamic nature of traffic network, and such updates always greatly affect the performance of route planning. Most existing methods, however, cannot effectively support traffic aware route planning. In this paper, a new strategy is proposed to handle this problem. We analysis the traffic condition on the road network and explore spatial-temporal knowledge to guide effective route planning. In particular, a set of effective techniques are used to avoid both unnecessary calculations on huge graph and excessive re-calculations caused by traffic condition updates. A comprehensive experiment is also conducted to evaluate the strategy performances.

1 Introduction

Location based services is growing in popularity in recent years. Many online route planning services such as Google Maps and Microsoft MapPoint have become one of the most important tools for our life nowadays. In addition, the growing popularity of location based services such as GPS navigation and logistic control has led to great interests in real time route planning techniques. Existing works [7, 9, 14] mainly focuses on the static route planning under current road condition. In this paper, we provide a novel approach to support traffic aware route planning in the presence of dynamic road networks through continuous monitoring.

As the road condition changes all the time, traffic aware route planning services is therefore very useful to most users. Most existing earliest arrival route planning strategies [11, 12] are simply based on the static road condition. However, in reality, the dynamics of road conditions should be considered in most cases. For example, a best route is planned for a driver at 6:30PM as Fig. 1(a), but this route becomes sub-optimal when he drives to point C at 7:00PM due to the road condition changes. At this time,

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 576–591, 2012.
© Springer-Verlag Berlin Heidelberg 2012

traffic-aware route planning is highly expected to adjust the route as Fig. 1 (b) for guiding users to bypass new congestions. We target to handle this problem in this paper to help users who tend to get stuck in the peak hour traffic congestions.

The major challenge comes from the expensive computational overhead caused by the continuous updates on the huge search graph, especially considering the in-efficiency of A* algorithm on time-based path search. On the Boston road network that contains over 40, 000 links [8], even a single calculation for earliest arrival route could cost iCarTel (an iphone application) several seconds. It is obviously not realistic to re-plan the route by A* search for each road condition update, especially when the number of queries goes up. As such, it is crucial to improve classical algorithms, by reducing the space (which contains relative edges) for each route search and avoiding re-computation caused by the dynamics of road condition. Due to the huge complexity of this problem, we seek to explore near-optimal solutions in an efficient and practical manner.

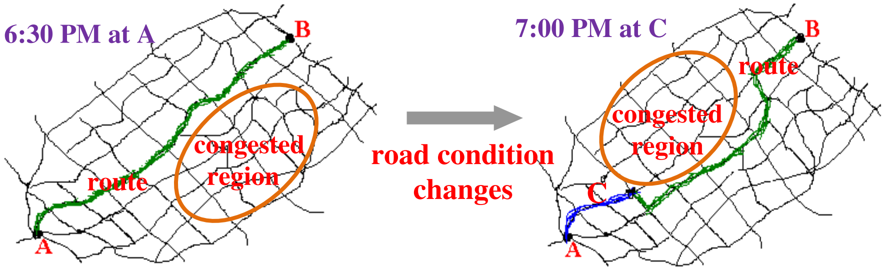


Fig. 1. (a) Initial earliest arrival route **(b)** Revised earliest arrival route

In this paper, an incremental route planning approach is proposed to achieve efficient traffic aware route planning. Particularly, a partial route (e.g. from A to C in Fig 1(b)), rather than the whole path from start to end, is computed each time. This approach enables the computational overhead to be greatly reduced for two main reasons: firstly, excessive re-calculations (particularly on far-away road segments) due to frequent road condition updates can be avoided; secondly, long distance route queries can be efficiently processed because each partial route search is restricted in a small region. Furthermore, a set of graph reduction and filtering mechanisms are used to improve algorithm efficiency, and issues like driving flexibility and congestion evolution are considered to improve planning effectiveness under dynamics. As such, the contributions in this paper are summarized as follows:

- We achieve traffic aware route planning. By adapting to the traffic condition changes, our approaches can guide drivers to bypass the new congestions and to follow the best route in dynamic road networks continuously.
- We develop a set of novel approaches to process queries in a small sub-graph each time. Computational overhead is significantly reduced to ensure real time feedback of route queries particularly for long distance queries.

- We take into account issues like congestion evolution and flexibility of path selection in driving. It contributes to achieve reliable route planning in presence of high dynamics of traffic condition in peak hour.

The rest of this paper is organized as follows. We discuss related work in Section 2. We specify the model and problem in Section 3, introduce the preliminaries in Section 4, and then present our query processing strategy in Section 5. We evaluate our solution using real data in Section 6 and conclude in Section 7.

2 Related Work

Route planning is a classical research problem and the related works can be generally divided into three categories: the first category is distance based shortest path, which aims to find the path that the distance is minimal; another category is time-dependent shortest path, which considers the actual speed on the roads and enable users to arrive destination as soon as possible; the last category is skyline search, which seeks to find skyline paths according to criterion like time, distance and high way cost.

Many efforts have been made to handle the distance based shortest path search problem. Dijkstra is a well known algorithm that finds shortest path on graph, and A* [13, 15] further improves the search performance by using heuristics. Also, a path indexing technique is proposed in [9, 10] to facilitate route search, and [4] presents an approach for handling probabilistic based shortest path query. Fast approximation algorithms is proposed in [7] to process route queries constrained by passing at least one from each category of places (e.g. to buy a book from a close book shop). The problem of skyline path query is also studied in works [1, 6] based on different criterion. Different from these works, we target to find the earliest arrival path by considering road conditions.

As the road condition is an essential factor that should be considered in practical route planning, increasing attentions have been put on time-dependent shortest path search in recent years. Work [2] studies how to answer queries of finding the best departure time that minimizes the total travel time from a place to another, over a road network, according to the average road condition in different time intervals derived from historical data. Also, a critical-time-point approach is presented in [3] to generate best routes and their corresponding time intervals, and the issue of finding fastest paths on a road network with speed patterns is discussed in [5]. However, these pattern-based approaches cannot support real time traffic-aware route planning.

In particular, work [8] targets to the problem answering continuous route planning queries over a road network in presence of updates to the delay estimates of links. Its basic idea is to calculate top-K (efficient) paths between any two vertexes according to statistics (simply average speed) on historical data at the build time, and then to keep ranking the top-K paths from current position to destination at the run time. However, it would cost huge space to store top-K paths between any two intersections, and the top-K paths cannot well balance different case features, e.g. the top-K paths in the raining, snow or sunny days could vary greatly. Instead, we assume the traffic is pure dynamic, and propose a new strategy to avoid un-necessary re-calculation caused by the road condition changes. Based on our approach, route search queries can be efficiently processed, and the dynamics of road network can be tracked to achieve traffic-aware route planning.

3 Traffic Aware Route Planning

In this section, we give a formal description of traffic aware route planning. A set of definitions based on the traffic network are presented to specify the problem we address in this paper:

Definition 1 (Dynamic road network): A road network is defined as a directed graph $G = <V, E>$, where $V = \{v\}$ is the set of vertices representing road ends or intersections, and $E = \{(v_i, v_j)|v_i, v_j \in V\}$ is the set of (directed) edges representing road segments. For each edge $e = (v_i, v_j) \in E$, the weight of this edge is the estimated time for passing through this edge $t(v_i, v_j)$ based on the current speed $s(e)$ on e. As the actual road speed keeps changing, the weight (i.e. time) of edges in G fluctuates from time to time accordingly, and G is thus a dynamic graph essentially.

Definition 2 (Vertices in/out-degree): Assume v is a vertex on a road network G, we use $ind(v)$ and $outd(v)$ to represent the in-degree and out-degree of v. Given a route r with set $r.V$ (all vertices it passes), the in-degree of r is defined as $IND(r) = \sum_{v \in r.V} ind(v)$ and out-degree is defined as $OUTD(r) = \sum_{v \in r.V} outd(v)$.

The in-degree and out-degree of vertices are important factors for route planning. Given a vertex v on the road network, congestions are more likely to occur on v if its in-degree value is high because more vehicles could move toward this interaction from different road segments. We thus tend to avoid such vertices. In contrast, vertices with high out-degree value are preferred because drivers have more (outgoing) paths to choose. Under a traffic network that is full of dynamics in the peak hour, such flexibility is useful for exception handling, e.g. when the previous reported route has to be adjusted to bypass new congestion or to switch to a light path.

Definition 3 (Route Query): In a dynamic road network G, a route query is defined as $qry=<src, dst>$, where src is the source node and dst is the destination node specified by users. We use this format as standard query type in this paper. In our system, users can also input $<src, dst,$ ”no”> to support fastest route query without continuous monitoring.

Definition 4 (Congested Region): Congested region is an area where drivers are very likely to get stuck for a long time if they come in. In the dynamic traffic network, this region must be stable (keep congested for certain duration). Otherwise, it would be meaningless for route planning. To ensure congestion regions been stable, a congested region is defined as CR = <CV, CE>, where CV is the involved (inside) set of vertices and $CE = \{(v, v') \mid v, v' \in CV\}$ is the involved edge set. In particular, it meets the following two constraints:

(1) This region is congested. For each road segment (edge) $e \in CE$, its speed must satisfies $s(e) < SP_{min}$, where SP_{min} is the minimum speed of normal travel and is set as 5km/h in this paper. That means, moving on the road segments in this region would be very slow;
(2) Large scale congested region. The size of this region satisfies $|CV| \geq SZ_{min}$ (SZ_{min}=4 in this paper). Large scale congested region means it is worthy to be noticed and congestions in this region are likely to be stable in certain duration.

Problem Definition. We target to solve the problem of traffic aware route planning which is formally defined as: Given a route query qry =<src, dst> on road network G = <V, E>, we process the query for a continuous optimal path (route) $r = (v_s, \text{v}, v', \ldots, v_t)$ on this dynamic road network that satisfies the following spatial-temporal optimisation goals and constraints: 1. Spatial constraints $v_s = src$ and $v_t = dst$; 2. Optimisation goal (shortest travel time) $\forall$ route $r' = (src, \ldots, dst)$: $T(r) \leq T(r')$. This problem is computationally hard because of the large graph it faces (road network) and the continuous recalculation due to updates on this graph (traffic condition). However, GPS navigation requires efficient query processing for immediate response. Therefore, an efficient approach is highly sought after to handle this problem.

4 Preliminary

In this section, we review the preliminaries of shortest path search, graph reduction and road condition monitoring in 4.1, 4.2 and 4.3 respectively. Effective shortest path search and graph reduction are essential issues of route planning.

4.1 Shortest Path Search (A* algorithm)

Shortest path search on graph is a classical topic and Dijkstra algorithm is a fundamental algorithm commonly used to solve it. Many approaches were designed to improve Dijkstra algorithm, and the most famous one is A* algorithm. It adopts the hill climbing method, and the ordering of search is based on the function $f(x) = g(x) + h(x)$, where $g(x)$ denotes the exact distance from starting node to current node x and $h(x)$ is an estimated distance (also the strict upper bound of the distance) to end node.

A* algorithm is usually adopted to find the shortest path on a static (current) graph. To avoid over-estimation, the distance and time used for computing heuristic estimation $h(x)$ are ‘as-the-crow-flies’ distance and maximum speed limit on the road segments respectively. This approach costs $O(m+n\log n)$ time, where m is the number edge and n is the number of vertices. However, A* algorithm is not effective for time-dependent shortest path cases because maximal speed limit is used in estimation (to guarantee $h(x)$ is not over-estimated). As A* heuristic is not sensitive to the actual road condition, earliest arrival query processing on large graph turns to be very in-efficient, particularly when the traffic is heavy. To ensure that time-dependent queries can be efficiently responded, our approach seeks to reduce the size of n, so that a route query can be processed in a small sub-graph each time to improve efficiency.

4.2 Graph Reduction

Though A* algorithm is very effective for shortest distance query, its performance is usually poor for earliest arriving route queries on a large graph. This is caused by the fact that traffic conditions cannot be used in the A* heuristics to guide route search. Also, the continuous monitoring on the dynamic traffic network must be restricted in a relative small sub-graph. Therefore, graph reduction techniques can be used to ensure that route queries can be efficiently processed.

Graph reduction is applied in [8] to facilitate shortest path algorithm for dynamic transportation networks. It reduces the whole space to an eclipse region, and then monitors the road condition change in this region. Only road segments in this region are admissible for route planning. Selected route is updated when the change in this region reaches a threshold because it may not be optimal since then. However, these approaches are too mechanical because they only concern the locations of start node and end node. In Section 5, we propose a set of spatial-temporal feature based techniques to narrow down the search space for each computation.

4.3 Road Condition Monitoring

As the dynamics of traffic network is a core issue for location based service optimisation, making use of information about updates of road condition in a close area is essential for achieving traffic-aware route planning. Many previous efforts have been endeavoured to handle this problem. Among the existing approaches, the most practical solution is the technique used in [8], which set up a relative region first, and then checks if the number of segments with delay updates that lie inside the region for a route query exceeds the average number of segments lying inside this region. If it is true, the re-run of the shortest path computation (by A* algorithm) is invoked then. In this paper, we adopt this basic idea, and propose an efficient monitoring approach to guarantee the selected route can be updated in a proper timing.

5 Efficient Path Query Processing Strategy

We present a novel strategy for traffic aware route planning in this section. In 5.1, we apply basic graph reduction and conduct congested region clustering as initialization to facilitate route planning. Then we select the top-k intermediate destinations according to a set of spatial-temporal criterion in 5.2. Afterwards in 5.3, route planning is conducted towards the top-k intermediate destinations. In this way, we can avoid re-calculation on far away road segments due to high possibility of speed change. By considering issues like spread of congestions, re-calculations impacted by the dynamics of road condition can be further reduced. We introduce the monitoring technique for adaptation of route planning to dynamic road conditions in 5.4 and present the main algorithm in 5.5.

5.1 Initialization

In initialization phase, basic graph reduction is conducted to ensure route search operations to be executed on a small sub-graph. Afterwards, congested regions are clustered. As congestion may spread and affect nearby areas, road segments close to the congested regions are identified. The spread of congestions onto these road segments will be considered to cope with the dynamics of traffic condition.

First of all, we conduct a basic graph reduction to settle a small sub-graph as the region relative to route query processing: an eclipse region *G'* shown in Fig. 2 (a) is efficiently derived from the whole space *G* in the same way as [8]. Only the road segments in this region are considered as relevant to route planning. However, as only positions of start and end nodes are considered, this eclipse region is very likely to be over-sized, and we only use it to set a base for route planning operations used in remaining sections.

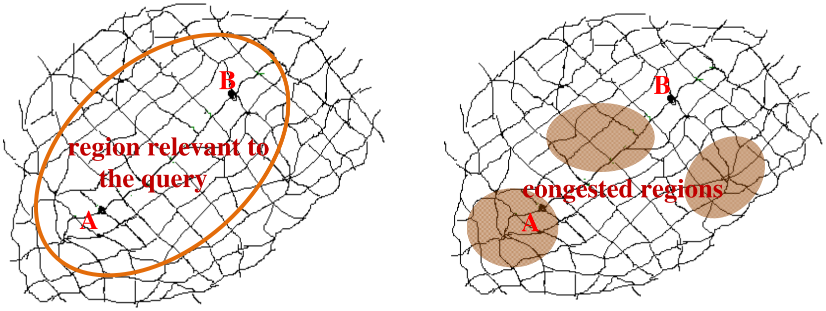


Fig. 2. (a) Basic graph reduction (eclipse region) **(b)** Congested region detection (clusters)

Then we make use of traffic report to generate congested regions (defined in Section 2) that are shown in Fig 2 (b). By doing so, smart route planning can be achieved by bypassing these regions and areas that may be affected. A congested region alerts us that this is an area drivers may get stuck for a long time. All congested regions in relative sub-graph *G'* are identified and saved into set *CRS* by clustering congested edges as follows: firstly, scan all edges in *G'* to generate congested edge set $CES = \{e \mid e \in G'.E \wedge s(e) < SP_{min}\}$, in which speed on each edge is less than a upper limit SP_{min}. Secondly, we find clusters in congested edges. Starting from the unprocessed edge in *CES* with minimal speed, expansion toward linking edges which belongs to *CES* is conducted iteratively. This region is validated and added to congested region set *CRS* if the number of involved vertices in this region is over SZ_{min}.

Further, the impact from congested regions onto nearby areas is estimated to deal with road condition changes and hence reduce re-computations. The spread of congestions always cause new jams on affected roads, and congestion spread is greatly impacted by traffic flow evolution (moving direction of cars in this flow). In reality, traffic flow evolution patterns can be easily derived by statistics. For example, 30% of cars on edge e_a go ahead on vertex *v*, 60% cars turn right to edge e_b and 10% turn left. We use function $flow(e_a, e_b)$ to represent the percentage of flow evolution from e_a to e_b

(i.e. 60%) according to historical data. Congestion on e_a is likely to spread to e_b if this percentage is high, e.g. 60%. Assume AS denotes the set containing all congested edges adjacent to edge e, we use the following formula to estimate the future speed on e that is likely to be affected by congested region:

$$estm\_s(e) = \prod_{e' \in AS} 1 - flow(e', e)^{1/4} \times s(e)$$

We check all edges linked from an node in congested region within 10 km distance and compute $estm\_s(e)$ for each of them. Speed is expected to decrease on e in the near future if many vehicles on a congested road segment e' will move to e. If e becomes congested, edges linked to it will be examined as well. Congested edges and estimated speed are updated periodically to ensure validation. Notice, this estimated speed only effects in a period, we use it in route planning if the time from current position to e based on current road condition is in 20 minutes.

5.2 Top-*k* Intermediate Destinations

In traditional approaches, a path strictly from starting point to destination is usually planned in each time. However it is not effective for earliest arriving route search because the frequent updates on road condition are likely to cause excessive re-calculation, particularly on faraway road segments. For example, if congestion occurs on a road segment that is part of planned route, re-planning is needed to guarantee service quality. For efficiency purpose, it is thus reasonable to plan partial path in a limited scope, rather than plan the whole route. Re-calculations caused by dynamic road condition can be greatly reduced accordingly. To set the boundary of route search properly, we must select some intermediate destinations as shown in Fig 3 (a), toward which effective route planning are conducted afterwards.

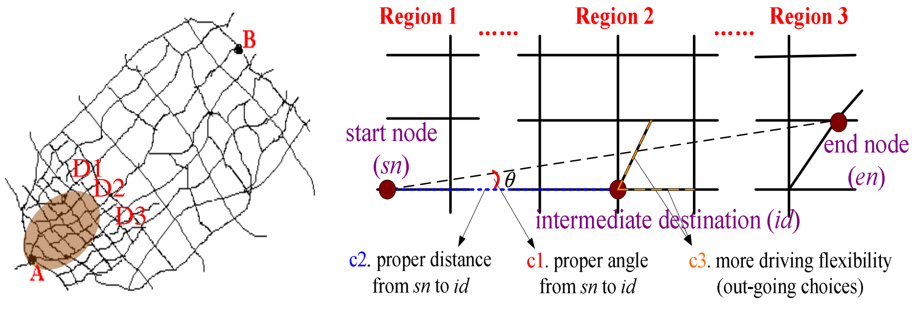


Fig. 3. (a) Intermediate destinations **(b)** Evaluation criterion

The selection of intermediate destinations must follow a set of spatial-temporal standards. First of all, the direction from starting point to intermediate destination has great evaluating merit. A good intermediate destination is expected to make the moving direction consistent with going toward final destination. Assume that $\theta((v_x, v), (v_x, v_d))$ is the angle of two straight lines ($v_s \rightarrow v_x$) and ($v_s \rightarrow v_d$), e.g. 'c1' in Fig. 3 (b), it can be seen as the difference between direction to intermediate destination and direction to final destination. We definitely prefer small angle because of less difference. To ensure the direction to be consistent, only angles in [0, $90^{\hbar}$] are accepted. Given that *cos*(*ang*) is the cosine value of an angle *ang*, which is in reverse proportion to the degree of *ang*. To evaluate the direction preference of selecting v_x as intermediate destination, we use $dw(v_x)$ to measure the direction weight of v_x as:

$$dw(v_x) = cos(\theta((v_s, v_x),(v_s, v_d)))$$

Meanwhile, the position of a vertex regarding to starting and end points is an important criterion according to 'c2' of Fig. 3 (b). A faraway intermediate destination may cause excessive re-calculation due to the frequent updates on traffic condition. Also, the distance should not be too close because the global view is neglected: it is hard to satisfy global optimisation when partial path search is made toward a intersection 200 meters away. To achieve a good balance between reducing re-calculation (not too faraway) and achieving global optimisation (not too close), we use position weight $pw(v_x)$ to measure intersectin v_x as intermediate destination as:

$$pw(v_x) = t_{max} - |h(v_s, v_x) - t_{best}|$$

where $h(v_s, v_x)$ denotes the average travel time for the 'as-the-crow-flies' distance (e.g. 10 miles) from v_s to v_x on the current issue time (e.g. 7PM). Statistical data is used here. T_{max} is the upper bound of $h(v_s, v_x)$ for pruning faraway vertices, and t_{best} is the time of favoured distance. We set t_{best} = 0.5 hour because experiments show that effective intermediate destinations tend to appear in region around 30-minutes-drive away. We also set t_{max}=1 hour to avoid re-calculations on faraway road segments, and thus $h(v_i, v_j) \in [0, 1]$ must be satisfied.

Another important criterion to evaluate intermediate destinations is the flexibility of future driving. According to 'c3' of Fig. 3(b), high flexibility means better capability for exception handling, e.g. to bypass new occurred congestions. Driving flexibility of an intermediate destination is determined by a set of spatial-temporal features. It is obvious that more out-going paths from an intersection give us more flexibility to choose. Among out-going edges, those in the same direction to final destination are definitely preferred. Wrapping up these issues, the flexibility weight $fw(v_x)$ for selecting vertex v_x as intermediate destination is calculated as:

$$fw(v_x) = \sum_{v \in FV(v_x)} cos(\theta((v_x, v),(v_x, v_d))/2)$$

where $FV(v_x) = \{v|\text{edge } e=(v_x, v) \in E\}$ is the forward vertices set of v_x, θ is the angle of straight lines ($v_x \rightarrow v$) and ($v_x \rightarrow v_d$). More out-going edges give drivers greater flexibility of route selection. For each out-going edge to v from v_x, we prefer the angle $angle((v_x, v), (v_x, v_d))$ to be small because its direction matches the required one. The cosine value is in reverse proportion to the angle size θ in range of $\theta \in$ [0, $90^{\hbar}$]. We thus use it to evaluate the preference of out-going paths regarding to moving direction.

To select proper intermediate destinations, issues mentioned above like spatial features and driving flexibility must be considered, as they benefit us to reduce re-calculations and to be more reliable under dynamics. In particular, the selectivity of intermediate destinations follows the following criterion:

$$w(v_x) = dw(v_x)/_{cd} + pw(v_x)/_{cp} + fw(v_x)/_{cf}$$

Where $dw(v_x)$ is the direction weight, $pw(v_x)$ is the position weight, $fw(v_x)$ is the flexibility weight. Factors *cd*, *cp* and *cf* represent the standard direction weight, position weight and flexibility weight required for capable intermediate destinations respectively. Thus, only vertices that satisfies $w(v)\geq 3$ are suitable as intermediate destination. By ranking $w(v)$ on vertices with $w(v)\geq 3$ not belonged to the congested region, K best vertices and final destination are selected as intermediate destination candidate into *IDC*. Route search is then conducted towards vertices in *IDC* for effective planning. If intermediate destination candidate cannot be detected, A* algorithm is simply used to search time-dependent shortest path to final destination v_t.

5.3 Route Search

Route search is made to find a path (patrial route) to one of the top-*k* intermediate destination. To find good result efficiently, we propose a novel algorithm to search the partial route with shortest time. From the start node to any of intermediate destination, we consider the exact time. From the intermediate destination to final destination, the estimated time is used to avoid traversing huge scale far-away road segments. Notice that it also contributes to reduce re-calculations caused by dynamics of road conditions on those road segments. Details of the (partial) route search toward those K intermediate destinations are given as follows.

```
Input:    $IDC$  –  intermediate destination candidates;
          $v_s$  –  starting node;
          $G'$   –  sub-graph of whole road network.
Output:   $pth$  –  a path (partial route).

$P = \{v_s\}$;                      // processed vertices
$A = \{v' | e = (v_s, v') \in G'.E'\}$  // vertices adjacent to $P$
$P' = \phi$ ;       // shortest path from $v_s$ to vertices in $P$
$lowB = 0$;       //lower bound of time to other vertices
$maxDiff = maxET(IDC, v_t) - minET(IDC, v_t)$;
do
  select $i, j$: min $TM(p_{si}) + t(v_i \in P, v_j \in A)$;
  $P = P + \{v_j\}$;
  $A = A + \{v' | (v_j, v') \in G'.E \wedge v' \notin P\} - \{v_j\}$;
  $p_{sj} = p_{si} + e(v_i, v_j)$;
  $P' = P' + \{(j, pth_{sj})\}$;
  $lowB = TM(p_{sj})$;
  if ($v_j \in IDC$) then insert $p_{sj}$ into $C$;   //path candidates
while ($\forall v_d \in IDC : TM(p_{sd}) + h(v_d, v_t) \geqslant lowB + maxDiff$)
$pth = selectFromPaths(C)$;
return $pth$;
```

Algorithm 1. Partial Path Search (PPS) Algorithm

Algorithm 1 is an efficient partial path search strategy. Compared with the conventional shortest path search, the optimisation here is made for the top-k rather than a single candidate, and the route returned is a partial one. It can be seen as an extension to the A* algorithm. Set P contains all the processed vertices, to which the earliest arrival time from starting node is known. Set A is contains vertices adjacent to P. That means, any vertex in A is not belonged to P, and there exist at least one in-coming edge from a node in P.

Based on a lower bound of total time to destination among processed vertices, we select an intersection v_j from A to which travel time is minimal (Line 7), then add it to P and update set A. The shortest path from starting node v_s to v_j is generated (Line 10) and recorded in path set P' (Line 11). The time from v_s to v_j becomes the lower bound of travel time from v_s to the remaining vertices $v \in G'.E/P$ (Line 12). If v_j is intermediate destination candidate, we further add this path into set C as a partial route candidate could be returned (Line 13). By keep doing so, we expand P in a way that time from start node is strictly increasing (Lines 6 – 14). This process can be when we successfully detect an intermediate destination v_d that has least value of $f(x) = TM(p_{sx})+h(v_x, v_t)$ in IDC in Line 14 (estimated speed can be used when necessary). Specifically, $f(x)$ is the total time, calculated as the sum of strict time $TM(p_{sx})$ from start node to v_x and the estimated time $h(v_x, v_t)$ from v_x to final destination. *maxDiff* is the maximal difference of this $h(v_x, v_t)$ value among vertices $v_x \in IDC$.

As the search process could continue when reaching intermediate destinations, we may need to select the best path among several path candidates stored in set C. The best path is selected by function *selectFromPaths*(C) in Line 15. Road structure features and estimated travel time of the paths are considered by this function. Firstly, we calculate the value of *OUTD(pth)/IND(pth)* for each path *pth*, and prune it if this value is not k best candidates and is less than a threshold. This guarantees the high driving flexibility and less time of route re-planning along the journey. Among those un-pruned paths, the one with minimal total travel time is returned as partial route.

5.4 Monitoring and Update

Due to the high dynamics of road condition in rash hour, it is essential to monitor the road condition and react to the relevant updates on it. Different from the previous approaches, road condition monitoring used in this paper is restricted in a relatively small region. In this way, computational overhead for the continuous monitoring can be significantly reduced.

Every time a (partial) route is planned, we start the monitoring phrase. Based on the starting and end vertices of the partial path derived from 5.3, an eclipse region is selected first for road condition monitoring in the same way as [8]. Then we start monitoring this region: for each arrival of road condition changes on the inside road segments, we compare its current speed with the speed when the partial path is generated. We update the partial path (route) if one of the following two conditions can be satisfied: (1) if the number of road segments on which the travel speed has been reduced is over a threshold a_1 (e.g. 80%), route updates should be conducted because a more efficient path is likely to be detected; (2) if the time of current partial route is significantly increased over a threshold a_2 (e.g. 15 minutes), it is very likely to find a new path that can reduce the travel time. When necessary, a new route from current position is calculated by carrying out algorithms 5.2 and 5.3.

5.5 Putting Them together

By integrating all above steps from 5.1 to 5.4, traffic aware earliest arrival route queries are processed in Algorithm 2.

```
Input:    G   –  road network;
          v_s –  current position.
Output:   r   –  planned route

 1  G' = r(G);
 2  do
 3    IDC = imd(G');
 4    p = PPS (IDC, v_s, G');
 5    if (incr_ratio(IDC) > a_1 or incr_time(p) > a_2)
 6      r = r + v(p);
 7      p = upd();
 8    else if  intermediate destination is reached
 9      r = r + path;
10    end if;
11  while (v_d is not reached)
12  return r;
```

Algorithm 2. Incremental Route Planning (IRP) Algorithm

In our Incremental Route Planning (IRP) algorithm (Algorithm 2), graph reduction and congestion spread evolution is conducted first according to section 5.1 (Line 1). Then, a set of intermediate destinations are selected as 5.2 (Line 3), and (partial) route planning toward the selected intermediate destinations is performed as 5.3 (Line 4). Then we monitor the road condition change (Line 5) according to 5.4. When necessary, the part of path *p* that driver has passed is added to route *r* (Line 6), and the partial path is re-calculated by calling intermediate destination selection and partial route search (Line 7). If the intermediate destination is reached, current path (user has passed it) is added to the travelled route (Line 9), and then we go back to Line 3 and start another round partial route search. This procedure continues until the final destination is reached.

6 Experimental Study

In this section, we experimentally evaluate the techniques that have been proposed in section 4. Specifically, our goal is to compare with existing approaches on: (1) the number of examined edges in a single query processing without considering road condition changes; (2) search accuracy, i.e. the actual travel time of the users along the whole trip; (3) the total time of the continuous query processing along the whole trip (consider dynamics of road condition).

A. Dataset and Experimental Setup

Our experiments were conducted on a HP Compaq 8180 Elite (i5 650) computer with 2-core CPUs at 3.2GHz and 1.12 GHz, 4GB RAM and running Windows XP operating system. We use a network graph that contains 37, 644 directed edges (road segments) and 28,342 vertices (intersections). Such map data corresponds to the road network of the city of Beijing, China, and they are stored with spatial grid indexes. As the actual speed on some road segments are not available, both real speed and simulation speed are used in this experiment. All experiments are based on 200 test cases and 100 cases with constant road speed and 100 cases with dynamic road speed.

B. Number of Examined Edges

Firstly, the number of examined edges for processing a single query of our method and existing approaches (Dijkstra and A*) is compared. Given that navigation centres have to handle massive requests in parallel, it is essential to reduce the number of edges examined for processing one query first for reducing computational overload. As Dijkstra and A* algorithms calculate the shortest distance on a fixed graph, dynamics of road condition is not considered. Particularly, we classify 100 test cases into groups according to average speed fist, and then compare the average number of examined edges for each group.

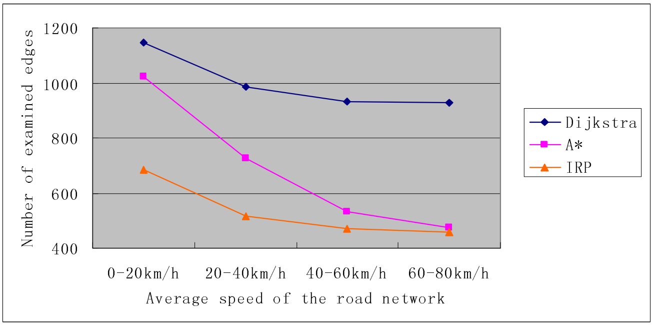


Fig. 4. Comparison of Examined Edges

Experimental results on the number of examined edges are shown as Fig 4. When the average road speed is between 0-20 km/h, the average numbers of examined edge of Dijkstra, A* and our IRP algorithm are 1145, 1021 and 684 respectively. For the test cases with average road speed in range of 20-40 km/h, the numbers of examined edges in average are reduced to 987, 725 and 517 respectively. When the average road speed is increased to a value between 60-80 km/h, the average (examined) edge number is further reduced to 926, 476 and 458 respectively.

From this figure, our approach is more superior to others when the traffic is heavy. IRP and A* tend to be have similar performance if the road condition is good. This reason is that excessive re-calculations can be avoided by IRP mechanism, particularly for the test cases with heavy traffic.

C. Travel Time on Planned Routes without Considering Road Speed Change

As the route returned by our IRP algorithm is a near optimal one, we compare the actual travel time of the IRP planned route with that of optimal route derived by A* search. To make them compatible, we assume the road condition to be static (constant road speed). Such comparison is also based on road speed classification.

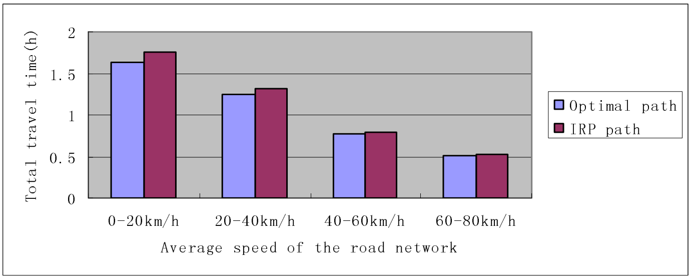


Fig. 5. Comparison of travel time to the optimal path

Experimental results on the travel time of selected routes are shown as Fig. 5. In the heavy traffic scenarios (0-20km/h), the average travel time of the path planned by IRP (1.75 hour) is 7.36% worse than the optimal route planned by A* (1.63 hour). In the second case (20-40km/h), their difference is reduced to 5.6%. Where road condition (40-80km/h) is good, the figure shows that the IRP planned route is almost optimal. Therefore, our approach can fundamentally find near-optimal routes, and the path planned by IRP is almost equivalent to the optimal result derived from A* search.

D. Total Processing Time Based on Dynamic Road Speed

To evaluate the algorithm efficiency under dynamic road network, we compare the processing time on single query processing which is adaptive to the dynamics of road speed. Such comparison is made between IRPS algorithm, pure Dijkstra and the revised

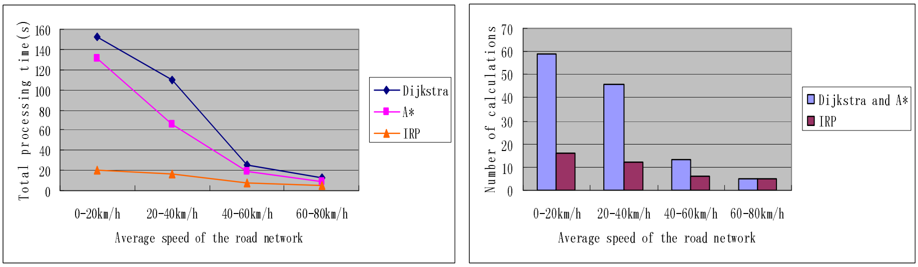


Fig. 6. (a) Processing time comparison **(b)** Comparison on calculation times

A* algorithm proposed in [8]. In addition, the total time of re-calculations is also compared for illustrating the benefit of intermediate destination selection.

For each route query, we conduct route planning continuously according to the road condition changes by different algorithms. Results of the average time of route updates and number of calculations for a single query based on different approaches are shown as Fig. 6 (a) and (b) respectively. According to Fig. 6 (a), the total processing time of IRP is less than Dijkstra and A*, and their difference is particularly huge when the traffic is heavy. It is obvious that IRP is the most efficient approach among three algorithms. From Fig. 6 (b), we can notice that route planning by IRP has much less time than Dijkstra and A* when the traffic is heavy. Therefore, the incremental approach adopted by IRP can significantly reduce the time of re-calculations, and IRP tend to support real time response because route search is restricted in a much smaller region.

7 Conclusion and Future Work

We presented a new strategy for achieve traffic aware route planning in this paper. As road condition is usually frequently updated, it is not realistic to plan the whole path every time due to the excessive re-planning operations. As such, an incremental planning approach is used. By selecting intermediate destinations, a partial path rather than whole path is planned each time for long distance queries. In this way, route planning is more efficient because it is carried out in a much smaller region, and un-necessary re-calculations caused by the dynamic road conditions can be avoided.

In the future, we will extract out and then use congestion evolution patterns through traffic data to further improve the performance of route planning.

Acknowledgement. This work was partially supported by Chinese NSFC project under grand numbers 91124001 and 60970030, and Australian ARC project under the grant number DP120102627.

References

[1] Deng, K., Zhou, X., Shen, H.T.: Multi-source skyline query processing in road networks. In: ICDE 2007, pp. 796–805. IEEE (2007)

[2] Ding, B., Yu, J.X., Qin, L.: Finding time-dependent shortest paths over large graphs. In: EDBT 2008, pp. 205–216. ACM (2008)

[3] Gunturi, V.M.V., Nunes, E., Yang, K., Shekhar, S.: A Critical-Time-Point Approach to all-Start-Time Lagrangian Shortest Paths: A Summary of Results. In: Pfoser, D., Tao, Y., Mouratidis, K., Nascimento, M.A., Mokbel, M., Shekhar, S., Huang, Y. (eds.) SSTD 2011. LNCS, vol. 6849, pp. 74–91. Springer, Heidelberg (2011)

[4] Hua, M., Pei, J.: Probabilistic path queries in road networks: traffic uncertainty aware path selection. In: EDBT 2010, pp. 347–358. ACM (2010)

[5] Kanoulas, E., et al.: Finding fastest paths on a road network with speed patterns. In: ICDE 2006, p. 10. IEEE (2006)

[6] Kriegel, H.-P., Renz, M., Schubert, M.: Route skyline queries: A multi-preference path planning approach. In: ICDE 2010, pp. 261–272. IEEE (2010)

[7] Li, F.-f., Cheng, D., Hadjieleftheriou, M., Kollios, G., Teng, S.-H.: On Trip Planning Queries in Spatial Databases. In: Medeiros, C.B., Egenhofer, M., Bertino, E. (eds.) SSTD 2005. LNCS, vol. 3633, pp. 273–290. Springer, Heidelberg (2005)

[8] Malviya, N., Madden, S., Bhattacharya, A.: A continuous query system for dynamic route planning. In: ICDE 2011, pp. 792–803. IEEE Computer Society (2011)

[9] Rice, M.N., Tsotras, V.J.: Graph indexing of road networks for shortest path queries with label restrictions. PVLDB 4(2), 69–80 (2010)

[10] Xiao, Y., et al.: Efficiently indexing shortest paths by exploiting symmetry in graphs. In: EDBT 2009, pp. 493–504. ACM (2009)

[11] Potamias, M., et al.: Fast shortest path distance estimation in large networks. In: CIKM 2009, pp. 867–876. ACM (2009)

[12] Gao, J., et al.: Fast top-k simple shortest paths discovery in graphs. In: CIKM 2010, pp. 509–518. ACM (2010)

[13] Likhachev, M., et al.: Anytime dynamic A*: An anytime, replanning algorithm. In: ICAPS 2005, pp. 262–271 (2005)

[14] Bast, H., et al.: In transit to constant time shortest-path queries in road networks. In: ALENEX (2007)

[15] Koenig, S., Likhachev, M., Furcy, D.: Lifelong planning A*. Artif. Intell. 155(1-2), 93–146 (2004)

Author Index